Algorithmic consequences of the Linearly Bounded Conjugator Property in braid groups

"Garside theory; state of the art and prospects" - Cap Hornu

Matthieu Calvez

Université Rennes 1/ Universidad de Sevilla

June 1st, 2012

Introduction

We consider the conjugacy problems in the braid groups B_n:

Introduction

We consider the conjugacy problems in the braid groups B_n:

$$B_n = \left\langle \sigma_1, \sigma_2 \ldots, \sigma_{n-1} : \begin{array}{cc} \sigma_i\sigma_j = \sigma_j\sigma_i & |i-j| \geqslant 2 \\ \sigma_i\sigma_{i+1}\sigma_i = \sigma_{i+1}\sigma_i\sigma_{i+1} & 1 \leqslant i \leqslant n-2 \end{array} \right\rangle.$$

Introduction

We consider the conjugacy problems in the braid groups B_n:

$$B_n = \left\langle \sigma_1, \sigma_2 \ldots, \sigma_{n-1} : \begin{array}{cc} \sigma_i\sigma_j = \sigma_j\sigma_i & |i-j| \geqslant 2 \\ \sigma_i\sigma_{i+1}\sigma_i = \sigma_{i+1}\sigma_i\sigma_{i+1} & 1 \leqslant i \leqslant n-2 \end{array} \right\rangle.$$

- CDP: Decide whether two given words represent conjugate elements.

Introduction

We consider the conjugacy problems in the braid groups B_n:

$$B_n = \left\langle \sigma_1, \sigma_2 \ldots, \sigma_{n-1} : \begin{array}{cc} \sigma_i\sigma_j = \sigma_j\sigma_i & |i-j| \geqslant 2 \\ \sigma_i\sigma_{i+1}\sigma_i = \sigma_{i+1}\sigma_i\sigma_{i+1} & 1 \leqslant i \leqslant n-2 \end{array} \right\rangle.$$

- CDP: Decide whether two given words represent conjugate elements.
- CSP: Find a conjugating element if there exists one.

Introduction

We consider the conjugacy problems in the braid groups B_n:

$$B_n = \left\langle \sigma_1, \sigma_2 \dots, \sigma_{n-1} : \begin{array}{cc} \sigma_i\sigma_j = \sigma_j\sigma_i & |i-j| \geqslant 2 \\ \sigma_i\sigma_{i+1}\sigma_i = \sigma_{i+1}\sigma_i\sigma_{i+1} & 1 \leqslant i \leqslant n-2 \end{array} \right\rangle .$$

- CDP: Decide whether two given words represent conjugate elements.
- CSP: Find a conjugating element if there exists one.

Both CDP and CSP are solvable in braid groups (Garside, 1969).

Search for a quick solution

- Garside's solution to CDP and CSP has high order of complexity, with respect to
  - the braid index n,
  - the length of the input l.

Search for a quick solution

- Garside's solution to CDP and CSP has high order of complexity, with respect to
  - the braid index n,
  - the length of the input l.
- General problem: find a quick solution, i.e. polynomial in n and l.

Search for a quick solution

- Garside's solution to CDP and CSP has high order of complexity, with respect to
  - the braid index n,
  - the length of the input l.
- General problem: find a quick solution, i.e. polynomial in n and l.
- Despite of many work in that direction : ElRifai-Morton, Birman-Ko-Lee, Birman-Gebhardt-González-Meneses, Gebhardt, Gebhardt-González-Meneses,...

Search for a quick solution

- Garside's solution to CDP and CSP has high order of complexity, with respect to
  - the braid index n,
  - the length of the input l.
- General problem: find a quick solution, i.e. polynomial in n and l.
- Despite of many work in that direction : ElRifai-Morton, Birman-Ko-Lee, Birman-Gebhardt-González-Meneses, Gebhardt, Gebhardt-González-Meneses,...
- the best known complexity remains exponential.

Search for a quick solution

- Garside's solution to CDP and CSP has high order of complexity, with respect to
  - the braid index n,
  - the length of the input l.
- General problem: find a quick solution, i.e. polynomial in n and l.
- Despite of many work in that direction : ElRifai-Morton, Birman-Ko-Lee, Birman-Gebhardt-González-Meneses, Gebhardt, Gebhardt-González-Meneses,...
- the best known complexity remains exponential.
- In this talk, n will be fixed and l will be the only parameter.

Search for a quick solution

A fast (polynomial) solution to CDP/CSP is known only in

Search for a quick solution

A fast (polynomial) solution to CDP/CSP is known only in

- $B_2 \cong \mathbb{Z}$,

Search for a quick solution

A fast (polynomial) solution to CDP/CSP is known only in

- $B_2 \cong \mathbb{Z}$,
- B_3.

Search for a quick solution

A fast (polynomial) solution to CDP/CSP is known only in

- $B_2 \cong \mathbb{Z}$,
- B_3.

Theorem (C., Wiest)

There is an algorithm for solving the CDP and CSP in the 4-strand braid group B_4 whose complexity depends cubically on the length of the input.

B_n as a Mapping Class Group

B_n as a Mapping Class Group

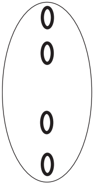
$$B_n \cong \mathcal{MCG}(\mathbb{D}_n, \partial\mathbb{D}_n).$$

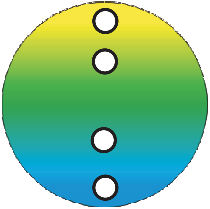
B_n as a Mapping Class Group

$$B_n \cong \mathcal{MCG}(\mathbb{D}_n, \partial\mathbb{D}_n).$$

B_n as a Mapping Class Group

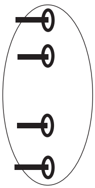
$$B_n \cong \mathcal{MCG}(\mathbb{D}_n, \partial\mathbb{D}_n).$$

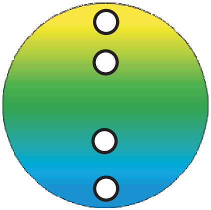




B_n as a Mapping Class Group

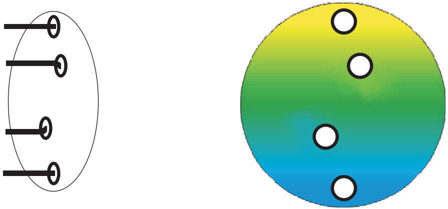
$$B_n \cong \mathcal{MCG}(\mathbb{D}_n, \partial\mathbb{D}_n).$$





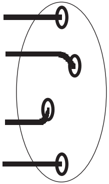
B_n as a Mapping Class Group

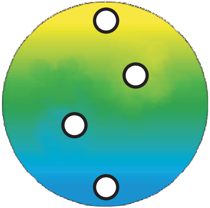
$$B_n \cong \mathcal{MCG}(\mathbb{D}_n, \partial\mathbb{D}_n).$$



B_n as a Mapping Class Group

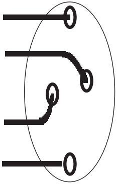
$$B_n \cong \mathcal{MCG}(\mathbb{D}_n, \partial\mathbb{D}_n).$$

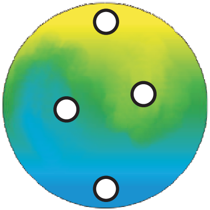




B_n as a Mapping Class Group

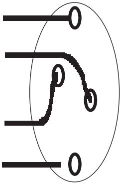
$$B_n \cong \mathcal{MCG}(\mathbb{D}_n, \partial\mathbb{D}_n).$$

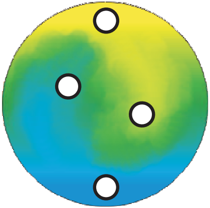




B_n as a Mapping Class Group

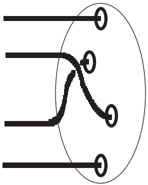
$$B_n \cong \mathcal{MCG}(\mathbb{D}_n, \partial\mathbb{D}_n).$$

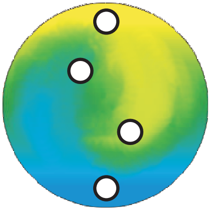




B_n as a Mapping Class Group

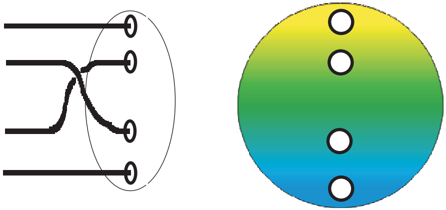
$$B_n \cong \mathcal{MCG}(\mathbb{D}_n, \partial\mathbb{D}_n).$$





B_n as a Mapping Class Group

$$B_n \cong \mathcal{MCG}(\mathbb{D}_n, \partial\mathbb{D}_n).$$



The Nielsen-Thurston trichotomy

Theorem (Thurston, 1980's)

Any braid $x \in B_n$ is either

- *periodic,*
- *pseudo-Anosov,*
- *or reducible non-periodic.*

The Nielsen-Thurston trichotomy

Theorem (Thurston, 1980's)

Any braid $x \in B_n$ is either

- *periodic,*
- *pseudo-Anosov,*
- *or reducible non-periodic.*

Reducible braids can be reduced in smaller "irreducible" pieces:

The Nielsen-Thurston trichotomy

Theorem (Thurston, 1980's)

Any braid $x \in B_n$ is either

- *periodic,*
- *pseudo-Anosov,*
- *or reducible non-periodic.*

Reducible braids can be reduced in smaller "irreducible" pieces: braids with less strands which are periodic or pA.

The Nielsen-Thurston trichotomy

Theorem (Thurston, 1980's)

Any braid $x \in B_n$ is either

- *periodic,*
- *pseudo-Anosov,*
- *or reducible non-periodic.*

Reducible braids can be reduced in smaller "irreducible" pieces: braids with less strands which are periodic or pA.

The Nielsen-Thurston type is invariant under conjugation and taking powers.

Use of N.-T. classification in the conjugacy problem

Idea: try to solve CDP and CSP in each of the particular types.

Use of N.-T. classification in the conjugacy problem

Idea: try to solve CDP and CSP in each of the particular types.

First, one needs to be able to decide quickly the Nielsen-Thurston type of a given braid

Use of N.-T. classification in the conjugacy problem

Idea: try to solve CDP and CSP in each of the particular types.

First, one needs to be able to decide quickly the Nielsen-Thurston type of a given braid
and for reducible braids, to find explicitly the decomposition into irreducible pieces.

Use of N.-T. classification in the conjugacy problem

Idea: try to solve CDP and CSP in each of the particular types.

First, one needs to be able to decide quickly the Nielsen-Thurston type of a given braid
and for reducible braids, to find explicitly the decomposition into irreducible pieces.

- In the periodic case, CDP and CSP were solved in $O(l^3 n^2 \log n)$ by Birman, Gebhardt and González-Meneses.

Use of N.-T. classification in the conjugacy problem

Idea: try to solve CDP and CSP in each of the particular types.

First, one needs to be able to decide quickly the Nielsen-Thurston type of a given braid
and for reducible braids, to find explicitly the decomposition into irreducible pieces.

- In the periodic case, CDP and CSP were solved in $O(l^3 n^2 \log n)$ by Birman, Gebhardt and González-Meneses.
- The pseudo-Anosov case is the one we will focus on.

Use of N.-T. classification in the conjugacy problem

Idea: try to solve CDP and CSP in each of the particular types.

First, one needs to be able to decide quickly the Nielsen-Thurston type of a given braid
and for reducible braids, to find explicitly the decomposition into irreducible pieces.

- In the periodic case, CDP and CSP were solved in $O(l^3 n^2 \log n)$ by Birman, Gebhardt and González-Meneses.
- The pseudo-Anosov case is the one we will focus on.
- In the reducible case, one can try to solve CDP and CSP by gluing "irreducible" pieces together.

The Linearly Bounded Conjugator Property

Theorem (Masur-Minsky, 2000)

Let n be a positive integer. Choose a generating set $\mathcal{G}_n$ for B_n. There exists a constant $C(\mathcal{G}_n)$ such that for any pair $x, y \in B_n$ of pseudo-Anosov conjugate braids, one can find a conjugator u between them satisfying

$$|u|_{\mathcal{G}_n} \leqslant C(\mathcal{G}_n) \cdot (|x|_{\mathcal{G}_n} + |y|_{\mathcal{G}_n}).$$

The Linearly Bounded Conjugator Property

Theorem (Masur-Minsky, 2000)

Let n be a positive integer. Choose a generating set $\mathcal{G}_n$ for B_n. There exists a constant $C(\mathcal{G}_n)$ such that for any pair $x, y \in B_n$ of pseudo-Anosov conjugate braids, one can find a conjugator u between them satisfying

$$|u|_{\mathcal{G}_n} \leqslant C(\mathcal{G}_n) \cdot (|x|_{\mathcal{G}_n} + |y|_{\mathcal{G}_n}).$$

Extended by Jing Tao (2011) to all Nielsen-Thurston types.

The Linearly Bounded Conjugator Property

Theorem (Masur-Minsky, 2000)

Let n be a positive integer. Choose a generating set $\mathcal{G}_n$ for B_n. There exists a constant $C(\mathcal{G}_n)$ such that for any pair $x, y \in B_n$ of pseudo-Anosov conjugate braids, one can find a conjugator u between them satisfying

$$|u|_{\mathcal{G}_n} \leqslant C(\mathcal{G}_n) \cdot (|x|_{\mathcal{G}_n} + |y|_{\mathcal{G}_n}).$$

Extended by Jing Tao (2011) to all Nielsen-Thurston types.

The constant C is NOT explicitly known.

Solving CSP/CDP in Garside groups

Solving CSP/CDP in Garside groups

Use of the Garside/ElRifai-Morton/Thurston normal form (greedy normal form).

Solving CSP/CDP in Garside groups

Use of the Garside/ElRifai-Morton/Thurston normal form (greedy normal form).

Define a special kind of conjugation, called “cyclic sliding” and denoted $\mathfrak{s}$ (Gebhardt & González-Meneses, 2008).

Some properties of the cyclic sliding operation

The cyclic sliding operation $\mathfrak{s}$

Some properties of the cyclic sliding operation

The cyclic sliding operation $\mathfrak{s}$

- simplifies normal forms: $length(\mathfrak{s}(x)) \leqslant length(x)$,

Some properties of the cyclic sliding operation

The cyclic sliding operation $\mathfrak{s}$

- simplifies normal forms: $\mathit{length}(\mathfrak{s}(x)) \leqslant \mathit{length}(x)$,
- is eventually periodic,

Some properties of the cyclic sliding operation

The cyclic sliding operation $\mathfrak{s}$

- simplifies normal forms: $length(\mathfrak{s}(x)) \leqslant length(x)$,
- is eventually periodic,
- has a small algorithmic cost.

Some properties of the cyclic sliding operation

The cyclic sliding operation $\mathfrak{s}$

- simplifies normal forms: $length(\mathfrak{s}(x)) \leqslant length(x)$,
- is eventually periodic,
- has a small algorithmic cost.

x

Some properties of the cyclic sliding operation

The cyclic sliding operation $\mathfrak{s}$

- simplifies normal forms: $length(\mathfrak{s}(x)) \leqslant length(x)$,
- is eventually periodic,
- has a small algorithmic cost.

Some properties of the cyclic sliding operation

The cyclic sliding operation $\mathfrak{s}$

- simplifies normal forms: $length(\mathfrak{s}(x)) \leqslant length(x)$,
- is eventually periodic,
- has a small algorithmic cost.

$$x \longrightarrow \mathfrak{s}(x)$$

Some properties of the cyclic sliding operation

The cyclic sliding operation $\mathfrak{s}$

- simplifies normal forms: $length(\mathfrak{s}(x)) \leqslant length(x)$,
- is eventually periodic,
- has a small algorithmic cost.

$$x \longrightarrow \mathfrak{s}(x) \longrightarrow \mathfrak{s}^2(x)$$

Some properties of the cyclic sliding operation

The cyclic sliding operation $\mathfrak{s}$

- simplifies normal forms: $length(\mathfrak{s}(x)) \leqslant length(x)$,
- is eventually periodic,
- has a small algorithmic cost.

$$x \longrightarrow \mathfrak{s}(x) \longrightarrow \mathfrak{s}^2(x) \longrightarrow \mathfrak{s}^3(x)$$

Some properties of the cyclic sliding operation

The cyclic sliding operation $\mathfrak{s}$

- simplifies normal forms: $length(\mathfrak{s}(x)) \leqslant length(x)$,
- is eventually periodic,
- has a small algorithmic cost.

$$x \longrightarrow \mathfrak{s}(x) \longrightarrow \mathfrak{s}^2(x) \longrightarrow \mathfrak{s}^3(x) \longrightarrow \cdots \longrightarrow$$

Some properties of the cyclic sliding operation

The cyclic sliding operation $\mathfrak{s}$

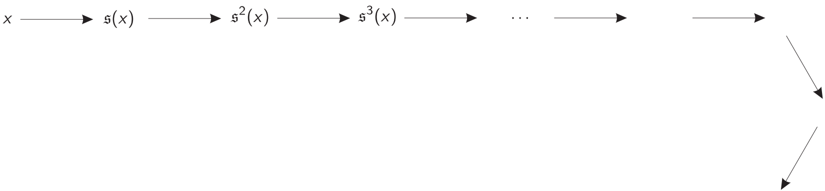
- simplifies normal forms: $length(\mathfrak{s}(x)) \leqslant length(x)$,
- is eventually periodic,
- has a small algorithmic cost.

$$x \longrightarrow \mathfrak{s}(x) \longrightarrow \mathfrak{s}^2(x) \longrightarrow \mathfrak{s}^3(x) \longrightarrow \cdots \longrightarrow \qquad \longrightarrow$$

Some properties of the cyclic sliding operation

The cyclic sliding operation $\mathfrak{s}$

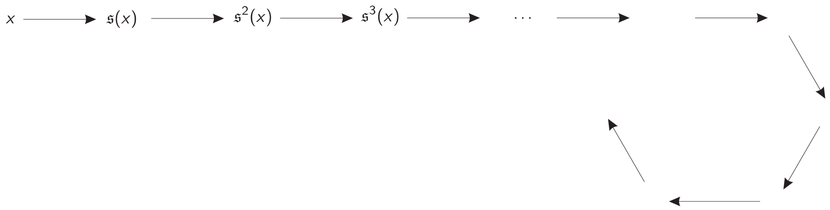
- simplifies normal forms: $length(\mathfrak{s}(x)) \leqslant length(x)$,
- is eventually periodic,
- has a small algorithmic cost.



Some properties of the cyclic sliding operation

The cyclic sliding operation $\mathfrak{s}$

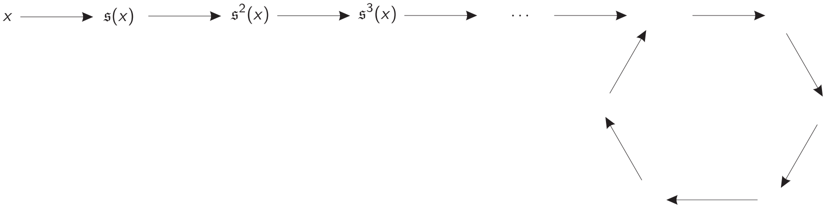
- simplifies normal forms: $length(\mathfrak{s}(x)) \leqslant length(x)$,
- is eventually periodic,
- has a small algorithmic cost.



Some properties of the cyclic sliding operation

The cyclic sliding operation $\mathfrak{s}$

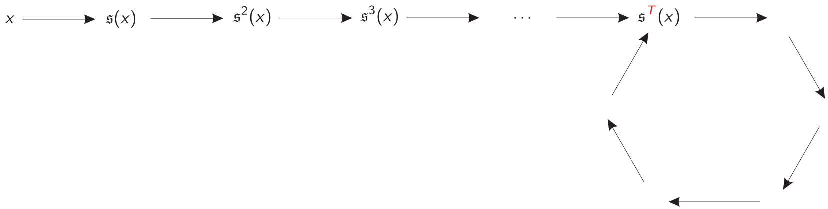
- simplifies normal forms: $length(\mathfrak{s}(x)) \leqslant length(x)$,
- is eventually periodic,
- has a small algorithmic cost.



Some properties of the cyclic sliding operation

The cyclic sliding operation $\mathfrak{s}$

- simplifies normal forms: $length(\mathfrak{s}(x)) \leqslant length(x)$,
- is eventually periodic,
- has a small algorithmic cost.

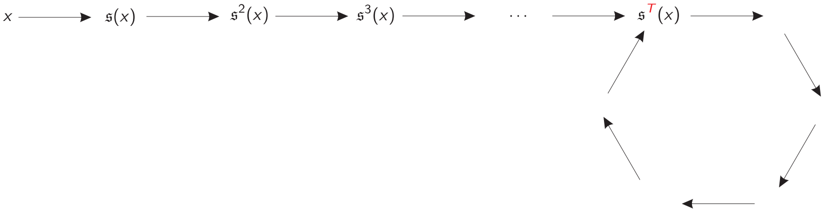


The set of sliding circuits

The set $\{y \in x^{B_n} \mid \exists k \in \mathbb{N}^* \mid \mathfrak{s}^k(y) = y\}$ is called the set of sliding circuits of x, denoted $SC(x)$.

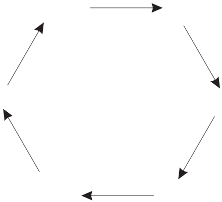
The set of sliding circuits

The set $\{y \in x^{B_n} \mid \exists k \in \mathbb{N}^* \mid \mathfrak{s}^k(y) = y\}$ is called the set of sliding circuits of x, denoted $SC(x)$.



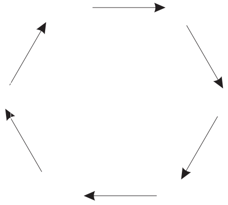
The set of sliding circuits

The set $\{y \in x^{B_n} \mid \exists k \in \mathbb{N}^* \mid \mathfrak{s}^k(y) = y\}$ is called the set of sliding circuits of x, denoted $SC(x)$.



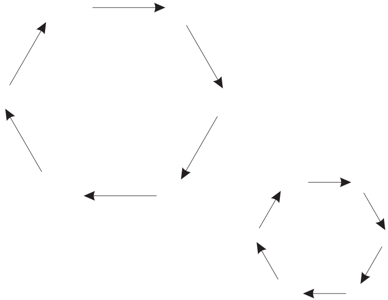
The set of sliding circuits

The set $\{y \in x^{B_n} \mid \exists k \in \mathbb{N}^* \mid \mathfrak{s}^k(y) = y\}$ is called the set of sliding circuits of x, denoted $SC(x)$.



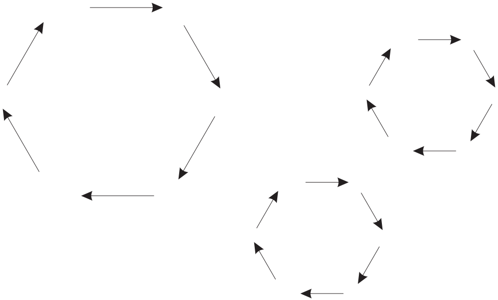
The set of sliding circuits

The set $\{y \in x^{B_n} \mid \exists k \in \mathbb{N}^* \mid \mathfrak{s}^k(y) = y\}$ is called the set of sliding circuits of x, denoted $SC(x)$.



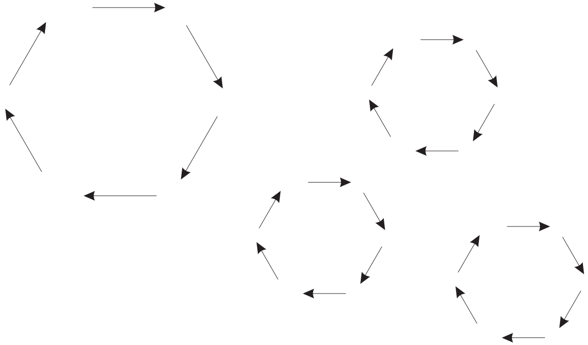
The set of sliding circuits

The set $\{y \in x^{B_n} \mid \exists k \in \mathbb{N}^* \mid \mathfrak{s}^k(y) = y\}$ is called the set of sliding circuits of x, denoted $SC(x)$.



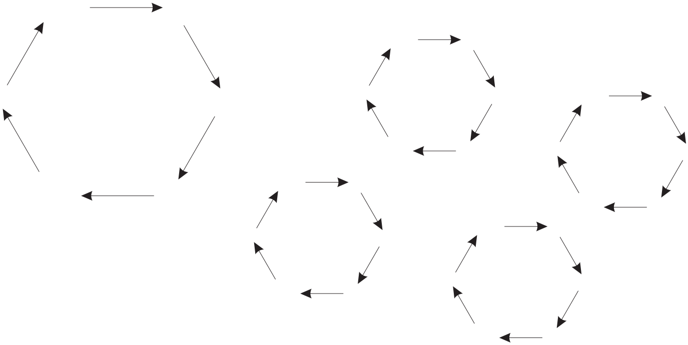
The set of sliding circuits

The set $\{y \in x^{B_n} \mid \exists k \in \mathbb{N}^* \mid \mathfrak{s}^k(y) = y\}$ is called the set of sliding circuits of x, denoted $SC(x)$.



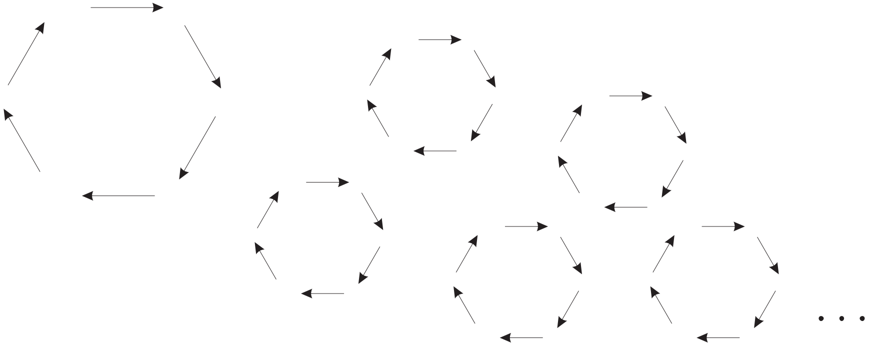
The set of sliding circuits

The set $\{y \in x^{B_n} \mid \exists k \in \mathbb{N}^* \mid \mathfrak{s}^k(y) = y\}$ is called the set of sliding circuits of x, denoted $SC(x)$.



The set of sliding circuits

The set $\{y \in x^{B_n} \mid \exists k \in \mathbb{N}^* \mid \mathfrak{s}^k(y) = y\}$ is called the set of sliding circuits of x, denoted $SC(x)$.



Some properties of $SC(x)$

- $SC(x)$ is a finite set,

Some properties of $SC(x)$

- $SC(x)$ is a finite set,
- for any $x, y \in B_n$, x and y are conjugate iff $SC(x) = SC(y)$.
  If not, $SC(x) \cap SC(y) = \emptyset$.

Some properties of $SC(x)$

- $SC(x)$ is a finite set,
- for any $x, y \in B_n$, x and y are conjugate iff $SC(x) = SC(y)$.
  If not, $SC(x) \cap SC(y) = \emptyset$.
- $SC(x)$ can be endowed with the structure of a directed connected graph (with edges given by “minimal conjugations”).

$\Rightarrow$ **this allows to solve CDP and CSP.**

Some properties of $SC(x)$

- $SC(x)$ is a finite set,
- for any $x, y \in B_n$, x and y are conjugate iff $SC(x) = SC(y)$.
  If not, $SC(x) \cap SC(y) = \emptyset$.
- $SC(x)$ can be endowed with the structure of a directed connected graph (with edges given by “minimal conjugations”).

$\Rightarrow$ **this allows to solve CDP and CSP.**

Some properties of $SC(x)$

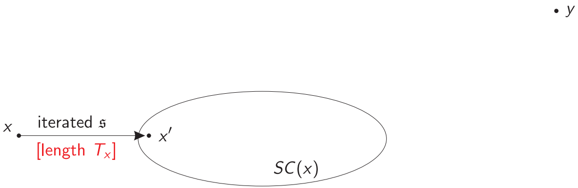
- $SC(x)$ is a finite set,
- for any $x, y \in B_n$, x and y are conjugate iff $SC(x) = SC(y)$.
  If not, $SC(x) \cap SC(y) = \emptyset$.
- $SC(x)$ can be endowed with the structure of a directed connected graph (with edges given by “minimal conjugations”).

$\Rightarrow$ **this allows to solve CDP and CSP.**

Some properties of $SC(x)$

- $SC(x)$ is a finite set,
- for any $x, y \in B_n$, x and y are conjugate iff $SC(x) = SC(y)$. If not, $SC(x) \cap SC(y) = \emptyset$.
- $SC(x)$ can be endowed with the structure of a directed connected graph (with edges given by "minimal conjugations").

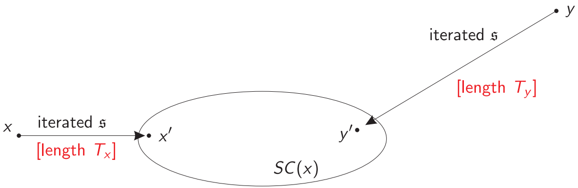
$\Rightarrow$ **this allows to solve CDP and CSP.**



Some properties of $SC(x)$

- $SC(x)$ is a finite set,
- for any $x, y \in B_n$, x and y are conjugate iff $SC(x) = SC(y)$. If not, $SC(x) \cap SC(y) = \emptyset$.
- $SC(x)$ can be endowed with the structure of a directed connected graph (with edges given by "minimal conjugations").

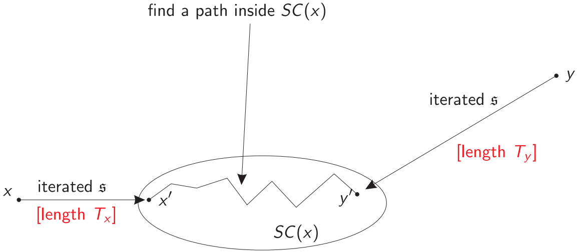
$\Rightarrow$ **this allows to solve CDP and CSP.**



Some properties of $SC(x)$

- $SC(x)$ is a finite set,
- for any $x, y \in B_n$, x and y are conjugate iff $SC(x) = SC(y)$.
  If not, $SC(x) \cap SC(y) = \emptyset$.
- $SC(x)$ can be endowed with the structure of a directed connected graph (with edges given by “minimal conjugations”).

$\Rightarrow$ **this allows to solve CDP and CSP.**



Two crucial questions

Two crucial questions

(1) Bound the size of $SC(x)$?

Two crucial questions

(1) Bound the size of $SC(x)$?

(2) Bound T (in terms of n and l)?

Two crucial questions

(1) Bound the size of $SC(x)$?

(2) Bound T (in terms of n and l)?
i.e. the number of iterations of $\mathfrak{s}$ needed before obtaining an element of SC

Two crucial questions

(1) Bound the size of $SC(x)$?

(2) Bound T (in terms of n and l)?
i.e. the number of iterations of $\mathfrak{s}$ needed before obtaining an element of SC

Hard to answer in general, a reason why we use the N.-T. classification.

Two crucial questions

(1) Bound the size of $SC(x)$?

(2) Bound T (in terms of n and l)?

i.e. the number of iterations of $\mathfrak{s}$ needed before obtaining an element of SC

Hard to answer in general, a reason why we use the N.-T. classification.

We want to answer in the pseudo-Anosov case.

The special case of rigid elements

Rigid braids are a special kind of element defined in terms of normal forms.

The special case of rigid elements

Rigid braids are a special kind of element defined in terms of normal forms.

The special case of rigid elements

Rigid braids are a special kind of element defined in terms of normal forms.

They satisfy:

The special case of rigid elements

Rigid braids are a special kind of element defined in terms of normal forms.

They satisfy:

- If x is rigid, then $\mathfrak{s}(x) = x$,

The special case of rigid elements

Rigid braids are a special kind of element defined in terms of normal forms.

They satisfy:

- If x is rigid, then $\mathfrak{s}(x) = x$,
- If x is conjugate to a rigid element then (Gebhardt, G.-Meneses)

$$SC(x) = \{\text{rigid conjugates of } x\}.$$

The special case of rigid elements

Rigid braids are a special kind of element defined in terms of normal forms.

They satisfy:

- If x is rigid, then $\mathfrak{s}(x) = x$,
- If x is conjugate to a rigid element then (Gebhardt, G.-Meneses)

$$SC(x) = \{\text{rigid conjugates of } x\}.$$

Moreover, rigidity is easy to check (just compute the normal form).

The special case of rigid elements

Rigid braids are a special kind of element defined in terms of normal forms.

They satisfy:

- If x is rigid, then $\mathfrak{s}(x) = x$,
- If x is conjugate to a rigid element then (Gebhardt, G.-Meneses)

$$SC(x) = \{\text{rigid conjugates of } x\}.$$

Moreover, rigidity is easy to check (just compute the normal form).

In general, the SC's of rigid elements have rather simple structure, although some difficulties may appear...

A (partial) solution to CDP/CSP for pA braids

A (partial) solution to CDP/CSP for pA braids

Theorem

If there exists a family of polynomials $P_n(l)$ *s.t. for any pA rigid n-braid* x, $\#SC(x) \leqslant P_n(length(x))$

A (partial) solution to CDP/CSP for pA braids

Theorem

If there exists a family of polynomials $P_n(l)$ s.t. for any pA rigid n-braid x, $\#SC(x) \leqslant P_n(length(x))$

then

A (partial) solution to CDP/CSP for pA braids

Theorem

If there exists a family of polynomials $P_n(l)$ s.t. for any pA rigid n-braid x, $\#SC(x) \leqslant P_n(length(x))$

then

there is an algorithm for solving CDP/CSP in the case of pA braids, whose complexity is polynomial in the length for any fixed n.

A (partial) solution to CDP/CSP for pA braids

Theorem

If there exists a family of polynomials $P_n(l)$ s.t. for any pA rigid n-braid x, $\#SC(x) \leqslant P_n(length(x))$

then

there is an algorithm for solving CDP/CSP in the case of pA braids, whose complexity is polynomial in the length for any fixed n.

In general, no polynomial bound (in l and n) on $\#SC$, for pA rigid braids (Prasolov).

Proof

Assumption $\Longrightarrow$ CDP/CSP polynomial (w.r.t. l for any fixed n) for rigid pA braids.

Proof

Assumption $\Longrightarrow$ CDP/CSP polynomial (w.r.t. l for any fixed n) for rigid pA braids.

Lemma

Given two pseudo-Anosov braids x and y, we can produce effectively $\bar{x}, \bar{y}$ pA rigid s.t.

- $x \sim y \Longleftrightarrow \bar{x} \sim \bar{y}$,
- *if so, the knowledge of a conjugator* $\bar{x} \longrightarrow \bar{y}$ *implies the knowledge of a conjugator* $x \longrightarrow y$,
- $length(\bar{x}) = O(length(x))$, $length(\bar{y}) = O(length(y))$.

The use of the linear bound

Theorem

Let x be a pseudo-Anosov braid. Suppose that x has some rigid conjugate. Then T_x is bounded above by $C \cdot \mathrm{length}(x)$. In particular, $\mathfrak{s}^{C \cdot \mathrm{length}(x)}(x)$ is rigid.

The use of the linear bound

Theorem

Let x be a pseudo-Anosov braid. Suppose that x has some rigid conjugate. Then T_x is bounded above by $C \cdot length(x)$. In particular, $\mathfrak{s}^{C \cdot length(x)}(x)$ is rigid.

Idea of proof:
Theorem (Gebhardt, G.-Meneses): If x has some rigid conjugate, then the shortest conjugating element from x to a rigid is to iterate cyclic sliding.

The use of the linear bound

Theorem

Let x be a pseudo-Anosov braid. Suppose that x has some rigid conjugate. Then T_x is bounded above by $C \cdot length(x)$. In particular, $\mathfrak{s}^{C \cdot length(x)}(x)$ is rigid.

Idea of proof:

Theorem (Gebhardt, G.-Meneses): If x has some rigid conjugate, then the shortest conjugating element from x to a rigid is to iterate cyclic sliding.

By Masur-Minsky, its length T is bounded by $C \cdot length(x)$. □

The use of the linear bound

Theorem

Let x be a pseudo-Anosov braid. Suppose that x has some rigid conjugate. Then T_x is bounded above by $C \cdot length(x)$. In particular, $\mathfrak{s}^{C \cdot length(x)}(x)$ is rigid.

Idea of proof:

Theorem (Gebhardt, G.-Meneses): If x has some rigid conjugate, then the shortest conjugating element from x to a rigid is to iterate cyclic sliding.

By Masur-Minsky, its length T is bounded by $C \cdot length(x)$. □

This gives a non-explicit linear bound on T above in the pA rigid case.

Passing to powers

We shall also use:

Passing to powers

We shall also use:

Theorem (Birman, Gebhardt, G.-Meneses)

For fixed n, there exists a (explicit) polynomial $K(n)$ s.t. for any pA n-braid x, there exists a power $m_x \leqslant K(n)$ with x^{m_x} conjugate to a rigid.

Main step

Theorem

There exists an algorithm of complexity $O(l^2)$ with:

- *INPUT: $x, y \in B_n$ pA (of length at most l),*
- *OUTPUT: $s \in \mathbb{N}$, $\bar{x}, \bar{y}, \widetilde{x}, \widetilde{y} \in B_n$ s.t.*

$$x^s \xrightarrow{\widetilde{x}} \bar{x} \qquad\qquad y^s \xrightarrow{\widetilde{y}} \bar{y}$$

with $\bar{x}$, $\bar{y}$ rigid, s bounded independently of length(x), length(y).

Proof of the first lemma

- $\bar{x}, \bar{y}$ pA rigid,

Proof of the first lemma

- $\bar{x}, \bar{y}$ pA rigid,
- $length(\bar{x}) = O(length(x))$, $length(\bar{y}) = O(length(y))$,

Proof of the first lemma

- $\bar{x}, \bar{y}$ pA rigid,
- $length(\bar{x}) = O(length(x)), length(\bar{y}) = O(length(y))$,
- $x \sim y \iff x^s \sim y^s \iff \bar{x} \sim \bar{y}$.

Proof of the first lemma

- $\bar{x}, \bar{y}$ pA rigid,
- $length(\bar{x}) = O(length(x)), length(\bar{y}) = O(length(y))$,
- $x \sim y \Longleftrightarrow x^s \sim y^s \Longleftrightarrow \bar{x} \sim \bar{y}$.

By unicity of roots of pA (G.-Meneses),

Proof of the first lemma

- $\bar{x}, \bar{y}$ pA rigid,
- $length(\bar{x}) = O(length(x))$, $length(\bar{y}) = O(length(y))$,
- $x \sim y \Longleftrightarrow x^s \sim y^s \Longleftrightarrow \bar{x} \sim \bar{y}$.

By unicity of roots of pA (G.-Meneses),

$$\Leftrightarrow$$

Proof of the first lemma

- $\bar{x}, \bar{y}$ pA rigid,
- $length(\bar{x}) = O(length(x))$, $length(\bar{y}) = O(length(y))$,
- $x \sim y \iff x^s \sim y^s \iff \bar{x} \sim \bar{y}$.

By unicity of roots of pA (G.-Meneses),

$$x^s \xrightarrow{\beta} y^s \quad \Leftrightarrow$$

Proof of the first lemma

- $\bar{x}, \bar{y}$ pA rigid,
- $length(\bar{x}) = O(length(x)), length(\bar{y}) = O(length(y))$,
- $x \sim y \iff x^s \sim y^s \iff \bar{x} \sim \bar{y}$.

By unicity of roots of pA (G.-Meneses),

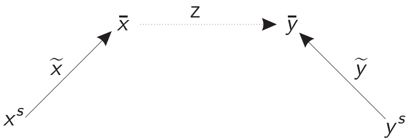
$$x^s \xrightarrow{\beta} y^s \Leftrightarrow x \xrightarrow{\beta} y$$

Proof of the first lemma

- $\bar{x}, \bar{y}$ pA rigid,
- $length(\bar{x}) = O(length(x))$, $length(\bar{y}) = O(length(y))$,
- $x \sim y \iff x^s \sim y^s \iff \bar{x} \sim \bar{y}$.

By unicity of roots of pA (G.-Meneses),

Finally, the previous algorithm also gives $\tilde{x}$ and $\tilde{y}$ s. t.



Description of the latter algorithm

Description of the latter algorithm

x

Description of the latter algorithm

x

x^2

Description of the latter algorithm

x

x^2

x^3

Description of the latter algorithm

x

x^2

x^3

$\vdots$

Description of the latter algorithm

x

x^2

x^3

$\vdots$

$x^{K(n)}$

Description of the latter algorithm

$$x \longrightarrow \mathfrak{s}(x)$$

$$x^2$$

$$x^3$$

$$\vdots$$

$$x^{K(n)}$$

Description of the latter algorithm

$$x \longrightarrow \mathfrak{s}(x)$$

$$x^2 \longrightarrow \mathfrak{s}(x^2)$$

$$x^3$$

$$\vdots$$

$$x^{K(n)}$$

Description of the latter algorithm

$$x \longrightarrow \mathfrak{s}(x)$$

$$x^2 \longrightarrow \mathfrak{s}(x^2)$$

$$x^3 \longrightarrow \mathfrak{s}(x^3)$$

$$\vdots$$

$$x^{K(n)}$$

Description of the latter algorithm

$$
\begin{array}{lcl}
x & \longrightarrow & \mathfrak{s}(x) \\
x^2 & \longrightarrow & \mathfrak{s}(x^2) \\
x^3 & \longrightarrow & \mathfrak{s}(x^3) \\
\vdots & & \\
x^{K(n)} & \longrightarrow & \mathfrak{s}(x^{K(n)})
\end{array}
$$

Description of the latter algorithm

$$x \longrightarrow \mathfrak{s}(x) \longrightarrow \mathfrak{s}^2(x)$$

$$x^2 \longrightarrow \mathfrak{s}(x^2)$$

$$x^3 \longrightarrow \mathfrak{s}(x^3)$$

$$\vdots$$

$$x^{K(n)} \longrightarrow \mathfrak{s}(x^{K(n)})$$

Description of the latter algorithm

$$\begin{array}{ccccc}
x & \longrightarrow & \mathfrak{s}(x) & \longrightarrow & \mathfrak{s}^2(x) \\
x^2 & \longrightarrow & \mathfrak{s}(x^2) & \longrightarrow & \mathfrak{s}^2(x^2) \\
x^3 & \longrightarrow & \mathfrak{s}(x^3) & & \\
\vdots & & & & \\
x^{K(n)} & \longrightarrow & \mathfrak{s}(x^{K(n)}) & &
\end{array}$$

Description of the latter algorithm

$$
\begin{array}{lllll}
x & \longrightarrow & \mathfrak{s}(x) & \longrightarrow & \mathfrak{s}^2(x) \\
x^2 & \longrightarrow & \mathfrak{s}(x^2) & \longrightarrow & \mathfrak{s}^2(x^2) \\
x^3 & \longrightarrow & \mathfrak{s}(x^3) & \longrightarrow & \mathfrak{s}^2(x^3) \\
\vdots & & & & \\
x^{K(n)} & \longrightarrow & \mathfrak{s}(x^{K(n)}) & &
\end{array}
$$

Description of the latter algorithm

$$\begin{array}{ccccc}
x & \longrightarrow & \mathfrak{s}(x) & \longrightarrow & \mathfrak{s}^2(x) \\
x^2 & \longrightarrow & \mathfrak{s}(x^2) & \longrightarrow & \mathfrak{s}^2(x^2) \\
x^3 & \longrightarrow & \mathfrak{s}(x^3) & \longrightarrow & \mathfrak{s}^2(x^3) \\
\vdots & & & & \\
x^{K(n)} & \longrightarrow & \mathfrak{s}(x^{K(n)}) & \longrightarrow & \mathfrak{s}^2(x^{K(n)})
\end{array}$$

Description of the latter algorithm

$$
\begin{array}{ccccccc}
x & \longrightarrow & \mathfrak{s}(x) & \longrightarrow & \mathfrak{s}^2(x) & \longrightarrow & \mathfrak{s}^3(x) \\
x^2 & \longrightarrow & \mathfrak{s}(x^2) & \longrightarrow & \mathfrak{s}^2(x^2) & \longrightarrow & \mathfrak{s}^3(x^2) \\
x^3 & \longrightarrow & \mathfrak{s}(x^3) & \longrightarrow & \mathfrak{s}^2(x^3) & \longrightarrow & \mathfrak{s}^3(x^3) \\
\vdots & & & & & & \\
x^{K(n)} & \longrightarrow & \mathfrak{s}(x^{K(n)}) & \longrightarrow & \mathfrak{s}^2(x^{K(n)}) & \longrightarrow & \mathfrak{s}^3(x^{K(n)})
\end{array}
$$

Description of the latter algorithm

$$
\begin{array}{ccccccccc}
x & \longrightarrow & \mathfrak{s}(x) & \longrightarrow & \mathfrak{s}^2(x) & \longrightarrow & \mathfrak{s}^3(x) & \longrightarrow & \\
x^2 & \longrightarrow & \mathfrak{s}(x^2) & \longrightarrow & \mathfrak{s}^2(x^2) & \longrightarrow & \mathfrak{s}^3(x^2) & \longrightarrow & \\
x^3 & \longrightarrow & \mathfrak{s}(x^3) & \longrightarrow & \mathfrak{s}^2(x^3) & \longrightarrow & \mathfrak{s}^3(x^3) & \longrightarrow & \\
\vdots & & & & & & & & \\
x^{K(n)} & \longrightarrow & \mathfrak{s}(x^{K(n)}) & \longrightarrow & \mathfrak{s}^2(x^{K(n)}) & \longrightarrow & \mathfrak{s}^3(x^{K(n)}) & \longrightarrow &
\end{array}
$$

Description of the latter algorithm

$$
\begin{array}{ccccccccc}
x & \longrightarrow & \mathfrak{s}(x) & \longrightarrow & \mathfrak{s}^2(x) & \longrightarrow & \mathfrak{s}^3(x) & \longrightarrow & \\
x^2 & \longrightarrow & \mathfrak{s}(x^2) & \longrightarrow & \mathfrak{s}^2(x^2) & \longrightarrow & \mathfrak{s}^3(x^2) & \longrightarrow & \\
x^3 & \longrightarrow & \mathfrak{s}(x^3) & \longrightarrow & \mathfrak{s}^2(x^3) & \longrightarrow & \mathfrak{s}^3(x^3) & \longrightarrow & \cdots \\
\vdots & & & & & & & & \\
x^{K(n)} & \longrightarrow & \mathfrak{s}(x^{K(n)}) & \longrightarrow & \mathfrak{s}^2(x^{K(n)}) & \longrightarrow & \mathfrak{s}^3(x^{K(n)}) & \longrightarrow &
\end{array}
$$

Description of the latter algorithm

$x \longrightarrow \mathfrak{s}(x) \longrightarrow \mathfrak{s}^2(x) \longrightarrow \mathfrak{s}^3(x) \longrightarrow$

$x^2 \longrightarrow \mathfrak{s}(x^2) \longrightarrow \mathfrak{s}^2(x^2) \longrightarrow \mathfrak{s}^3(x^2) \longrightarrow$

$x^3 \longrightarrow \mathfrak{s}(x^3) \longrightarrow \mathfrak{s}^2(x^3) \longrightarrow \mathfrak{s}^3(x^3) \longrightarrow$ $\cdots$ $\mathfrak{s}^T(x^{m_x})$ rigid!

$\vdots$

$x^{K(n)} \longrightarrow \mathfrak{s}(x^{K(n)}) \longrightarrow \mathfrak{s}^2(x^{K(n)}) \longrightarrow \mathfrak{s}^3(x^{K(n)}) \longrightarrow$

Description of the latter algorithm

$$x \longrightarrow \mathfrak{s}(x) \longrightarrow \mathfrak{s}^2(x) \longrightarrow \mathfrak{s}^3(x) \longrightarrow$$

$$x^2 \longrightarrow \mathfrak{s}(x^2) \longrightarrow \mathfrak{s}^2(x^2) \longrightarrow \mathfrak{s}^3(x^2) \longrightarrow$$

$$x^3 \longrightarrow \mathfrak{s}(x^3) \longrightarrow \mathfrak{s}^2(x^3) \longrightarrow \mathfrak{s}^3(x^3) \longrightarrow \qquad \cdots \ \mathfrak{s}^T(x^{m_x}) \text{ rigid!}$$

$$\vdots$$

$$x^{K(n)} \longrightarrow \mathfrak{s}(x^{K(n)}) \longrightarrow \mathfrak{s}^2(x^{K(n)}) \longrightarrow \mathfrak{s}^3(x^{K(n)}) \longrightarrow$$

Description of the latter algorithm

$$
\begin{array}{ccccccccc}
x & \longrightarrow & \mathfrak{s}(x) & \longrightarrow & \mathfrak{s}^2(x) & \longrightarrow & \mathfrak{s}^3(x) & \longrightarrow & \\
x^2 & \longrightarrow & \mathfrak{s}(x^2) & \longrightarrow & \mathfrak{s}^2(x^2) & \longrightarrow & \mathfrak{s}^3(x^2) & \longrightarrow & \\
x^3 & \longrightarrow & \mathfrak{s}(x^3) & \longrightarrow & \mathfrak{s}^2(x^3) & \longrightarrow & \mathfrak{s}^3(x^3) & \longrightarrow & \cdots\ \mathfrak{s}^T(x^{m_x}) \text{ rigid!} \\
\vdots & & & & & & & & \\
x^{K(n)} & \longrightarrow & \mathfrak{s}(x^{K(n)}) & \longrightarrow & \mathfrak{s}^2(x^{K(n)}) & \longrightarrow & \mathfrak{s}^3(x^{K(n)}) & \longrightarrow &
\end{array}
$$

- Linear (not explicit) number of iterations of cyclic slidings (w.r.t. l).
-
-
-

Description of the latter algorithm

$$
\begin{array}{ccccccccc}
x & \longrightarrow & \mathfrak{s}(x) & \longrightarrow & \mathfrak{s}^2(x) & \longrightarrow & \mathfrak{s}^3(x) & \longrightarrow & \\
x^2 & \longrightarrow & \mathfrak{s}(x^2) & \longrightarrow & \mathfrak{s}^2(x^2) & \longrightarrow & \mathfrak{s}^3(x^2) & \longrightarrow & \\
x^3 & \longrightarrow & \mathfrak{s}(x^3) & \longrightarrow & \mathfrak{s}^2(x^3) & \longrightarrow & \mathfrak{s}^3(x^3) & \longrightarrow & \cdots\ \mathfrak{s}^T(x^{m_x}) \text{ rigid!} \\
\vdots & & & & & & & & \\
x^{K(n)} & \longrightarrow & \mathfrak{s}(x^{K(n)}) & \longrightarrow & \mathfrak{s}^2(x^{K(n)}) & \longrightarrow & \mathfrak{s}^3(x^{K(n)}) & \longrightarrow &
\end{array}
$$

- Linear (not explicit) number of iterations of cyclic slidings (w.r.t. l).
- Call $x' = \mathfrak{s}^T(x^{m_x})$. Do the same for y for obtaining y'.
-
-

Description of the latter algorithm

$$x \longrightarrow \mathfrak{s}(x) \longrightarrow \mathfrak{s}^2(x) \longrightarrow \mathfrak{s}^3(x) \longrightarrow$$

$$x^2 \longrightarrow \mathfrak{s}(x^2) \longrightarrow \mathfrak{s}^2(x^2) \longrightarrow \mathfrak{s}^3(x^2) \longrightarrow$$

$$x^3 \longrightarrow \mathfrak{s}(x^3) \longrightarrow \mathfrak{s}^2(x^3) \longrightarrow \mathfrak{s}^3(x^3) \longrightarrow \quad \cdots \ \mathfrak{s}^T(x^{m_x}) \text{ rigid!}$$

$$\vdots$$

$$x^{K(n)} \longrightarrow \mathfrak{s}(x^{K(n)}) \longrightarrow \mathfrak{s}^2(x^{K(n)}) \longrightarrow \mathfrak{s}^3(x^{K(n)}) \longrightarrow$$

- Linear (not explicit) number of iterations of cyclic slidings (w.r.t. l).
- Call $x' = \mathfrak{s}^T(x^{m_x})$. Do the same for y for obtaining y'.
- Take $s = lcm(m_x, m_y)$ and $\bar{x} = x'^{\frac{s}{m_x}}$, $\bar{y} = y'^{\frac{s}{m_y}}$.
-

Description of the latter algorithm

$$\begin{array}{ccccccccc}
x & \longrightarrow & \mathfrak{s}(x) & \longrightarrow & \mathfrak{s}^2(x) & \longrightarrow & \mathfrak{s}^3(x) & \longrightarrow & \\
x^2 & \longrightarrow & \mathfrak{s}(x^2) & \longrightarrow & \mathfrak{s}^2(x^2) & \longrightarrow & \mathfrak{s}^3(x^2) & \longrightarrow & \\
x^3 & \longrightarrow & \mathfrak{s}(x^3) & \longrightarrow & \mathfrak{s}^2(x^3) & \longrightarrow & \mathfrak{s}^3(x^3) & \longrightarrow & \cdots\ \mathfrak{s}^T(x^{m_x}) \text{ rigid!} \\
\vdots & & & & & & & & \\
x^{K(n)} & \longrightarrow & \mathfrak{s}(x^{K(n)}) & \longrightarrow & \mathfrak{s}^2(x^{K(n)}) & \longrightarrow & \mathfrak{s}^3(x^{K(n)}) & \longrightarrow &
\end{array}$$

- Linear (not explicit) number of iterations of cyclic slidings (w.r.t. l).
- Call $x' = \mathfrak{s}^T(x^{m_x})$. Do the same for y for obtaining y'.
- Take $s = lcm(m_x, m_y)$ and $\bar{x} = x'^{\frac{s}{m_x}}$, $\bar{y} = y'^{\frac{s}{m_y}}$.
- $\tilde{x}$, $\tilde{y}$ are obtained as the product of arrows involved in cyclic slidings.

Solution to CDP/CSP in B_4

-
-

Solution to CDP/CSP in B_4

- The problem of deciding the Nielsen-Thurston type of a given 4-strand braid has a quadratic solution (C.-Wiest).
-

Moreover:

Solution to CDP/CSP in B_4

- The problem of deciding the Nielsen-Thurston type of a given 4-strand braid has a quadratic solution (C.-Wiest).
- For 4-strands *reducible* braids, CDP and CSP are solvable by a fast algorithm (C.-Wiest).

Moreover:

Theorem (C., Wiest)

Let $x \in B_4$ be a pseudo-Anosov rigid braid. Then $\#SC(x)$ is bounded above by $O(l^2)$.

Solution to CDP/CSP in B_4

- The problem of deciding the Nielsen-Thurston type of a given 4-strand braid has a quadratic solution (C.-Wiest).
- For 4-strands *reducible* braids, CDP and CSP are solvable by a fast algorithm (C.-Wiest).

Moreover:

Theorem (C., Wiest)

Let $x \in B_4$ be a pseudo-Anosov rigid braid. Then $\#SC(x)$ is bounded above by $O(l^2)$.

Corollary

There is an algorithm of complexity $O(l^3)$ solving CDP/CSP in B_4.

Structure of SC's of rigid elements

Remark: We use the Birman-Ko-Lee structure.

Structure of SC's of rigid elements

Remark: We use the Birman-Ko-Lee structure.

The set of Sliding circuits is endowed with the structure of a connected directed graph.

Structure of SC's of rigid elements

Remark: We use the Birman-Ko-Lee structure.

The set of Sliding circuits is endowed with the structure of a connected directed graph.

The vertices are the elements of SC, the edges some “minimal conjugators”.

Structure of SC's of rigid elements

Remark: We use the Birman-Ko-Lee structure.

The set of Sliding circuits is endowed with the structure of a connected directed graph.

The vertices are the elements of SC, the edges some “minimal conjugators”.

In the rigid case, results by Gebhardt about the ”transport” allow to consider this graph modulo conjugation by δ and cycling.

Structure of SC's of rigid elements

Remark: We use the Birman-Ko-Lee structure.

The set of Sliding circuits is endowed with the structure of a connected directed graph.

The vertices are the elements of SC, the edges some "minimal conjugators".

In the rigid case, results by Gebhardt about the "transport" allow to consider this graph modulo conjugation by δ and cycling.

We obtain the quotient graph $SC_{\sim}(x)$:

Structure of SC's of rigid elements

Remark: We use the Birman-Ko-Lee structure.

The set of Sliding circuits is endowed with the structure of a connected directed graph.

The vertices are the elements of SC, the edges some "minimal conjugators".

In the rigid case, results by Gebhardt about the "transport" allow to consider this graph modulo conjugation by δ and cycling.

We obtain the quotient graph $SC_{\sim}(x)$:

- each vertex corresponds to an equivalence class containing $4 \cdot \mathit{length}(x)$ elements of $SC(x)$

Structure of SC's of rigid elements

Remark: We use the Birman-Ko-Lee structure.

The set of Sliding circuits is endowed with the structure of a connected directed graph.

The vertices are the elements of SC, the edges some "minimal conjugators".

In the rigid case, results by Gebhardt about the "transport" allow to consider this graph modulo conjugation by δ and cycling.

We obtain the quotient graph $SC_\sim(x)$:

- each vertex corresponds to an equivalence class containing $4 \cdot length(x)$ elements of $SC(x)$
- the edges still correspond to "minimal conjugators".

Structure of SC's of rigid elements

Remark: We use the Birman-Ko-Lee structure.

The set of Sliding circuits is endowed with the structure of a connected directed graph.

The vertices are the elements of SC, the edges some "minimal conjugators".

In the rigid case, results by Gebhardt about the "transport" allow to consider this graph modulo conjugation by δ and cycling.

We obtain the quotient graph $SC_{\sim}(x)$:

- each vertex corresponds to an equivalence class containing $4 \cdot length(x)$ elements of $SC(x)$
- the edges still correspond to "minimal conjugators".

We need to bound linearly (w.r.t. the length) the number of vertices of $SC_{\sim}(x)$.

The quotient graph

Thanks to the simplicity of the lattice of simple elements in the Birman-Ko-Lee structure of B_4, one can show that this quotient graph $SC_{\sim}(x)$ has one of the following forms.

The quotient graph

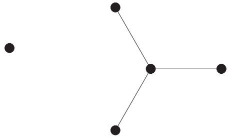
Thanks to the simplicity of the lattice of simple elements in the Birman-Ko-Lee structure of B_4, one can show that this quotient graph $SC_{\sim}(x)$ has one of the following forms.

The quotient graph

Thanks to the simplicity of the lattice of simple elements in the Birman-Ko-Lee structure of B_4, one can show that this quotient graph $SC_{\sim}(x)$ has one of the following forms.

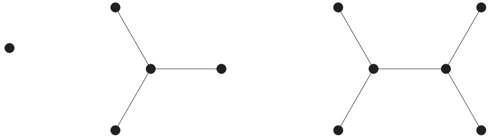
The quotient graph

Thanks to the simplicity of the lattice of simple elements in the Birman-Ko-Lee structure of B_4, one can show that this quotient graph $SC_{\sim}(x)$ has one of the following forms.



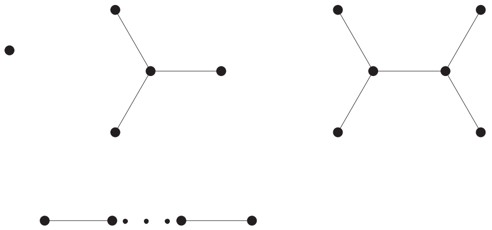
The quotient graph

Thanks to the simplicity of the lattice of simple elements in the Birman-Ko-Lee structure of B_4, one can show that this quotient graph $SC_{\sim}(x)$ has one of the following forms.



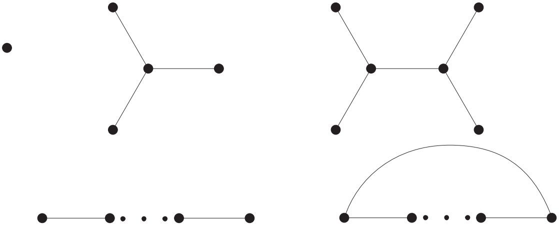
The quotient graph

Thanks to the simplicity of the lattice of simple elements in the Birman-Ko-Lee structure of B_4, one can show that this quotient graph $SC_{\sim}(x)$ has one of the following forms.



The quotient graph

Thanks to the simplicity of the lattice of simple elements in the Birman-Ko-Lee structure of B_4, one can show that this quotient graph $SC_{\sim}(x)$ has one of the following forms.



Bounding the line

As edges are given by minimal conjugators we can use again Masur-Minsky's bound: the length of the line is linearly bounded by the length of the braid we started with.

Another (not explicit) polynomial-time algorithm

Another (not explicit) polynomial-time algorithm

Theorem (C.)

There exists an algorithm which decides the Nielsen-Thurston type of a given braid on n strands and of length l in time $O(l^3)$ *for each fixed n.*

The algorithm

1) It is easy to decide periodicity (Birman, Gebhardt, G.-Meneses).

The algorithm

1) It is easy to decide periodicity (Birman, Gebhardt, G.-Meneses).
2) Pseudo-Anosov have small ($\leqslant K(n)$) power conjugated to a rigid braid.

The algorithm

1) It is easy to decide periodicity (Birman, Gebhardt, G.-Meneses).
2) Pseudo-Anosov have small ($\leqslant K(n)$) power conjugated to a rigid braid.
3) Given $x \in B_n$ non-periodic, for any $i = 1, \ldots, K(n)$, apply $\mathfrak{s}$ iteratively $C \cdot length(x^i)$ times to x^i.

The algorithm

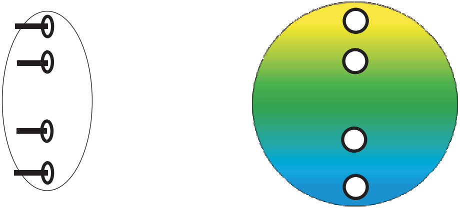
1) It is easy to decide periodicity (Birman, Gebhardt, G.-Meneses).
2) Pseudo-Anosov have small ($\leqslant K(n)$) power conjugated to a rigid braid.
3) Given $x \in B_n$ non-periodic, for any $i = 1, \ldots, K(n)$, apply $\mathfrak{s}$ iteratively $C \cdot \mathit{length}(x^i)$ times to x^i.
4) If no rigid element is found, then x is reducible.

The algorithm

1) It is easy to decide periodicity (Birman, Gebhardt, G.-Meneses).
2) Pseudo-Anosov have small ($\leqslant K(n)$) power conjugated to a rigid braid.
3) Given $x \in B_n$ non-periodic, for any $i = 1, \ldots, K(n)$, apply $\mathfrak{s}$ iteratively $C \cdot \mathit{length}(x^i)$ times to x^i.
4) If no rigid element is found, then x is reducible.
5) Otherwise, for the rigid element $\widetilde{x}$ obtained, one can test in an effective way whether it is reducible or pseudo-Anosov (G.-Meneses, Wiest).

Questions

- Look at the geometry of the curve complex associated to the n-times punctured disk and find the value of C.
- Does LBC hold in Garside groups?



[1]

Thank you

[1] This picture by courtesy of Marta Aguilera.