Lecture 9 – Mon Mar 13

Recall: Elrifai–Morton's normal forms for braids:

- define an order relation $A \le B$ if $\exists C_1, C_2 \in B_n^+$ s.t. $B = C_1 A C_2$
- $[r,s] = \{ B \,/\, \Delta^r \le B \le \Delta^s \}$, Δ = Garside elt

  $[0,1] = \{\text{permutation braids}\} =: S_n^+$

  $\inf(B) = \max\{ r \,/\, \Delta^r \le B \}$, $\sup(B) = \min\{ s \,/\, B \le \Delta^s \}$

Thm: $\forall P \in B_n$, $\exists!$ decomp. $P = \Delta^r A_1 \dots A_k$, $r \in \mathbb{Z}$, $A_i \in S_n^+ \setminus \{e, \Delta\}$

$S(A_{i+1}) \subset F(A_i)$ ← finishing set

$\inf B = r$, $\sup B = r+k$

starting set $= \{ k \,/\, A = \sigma_k A'$ for some $A' \in S_n^+ \}$

$= \{ k \,/\, \pi(k) > \pi(k+1) \}$

This normal form gives a solution to the word problem, given algorithm to compute it.

- start with expression $P = \Delta^r P'$, $P' \in B_n^+$ given as $B_1 \dots B_k$ (permutation braids)

  [e.g.: stupid way from expr. as $\prod \sigma_i^{\pm 1}$: replace each σ_i^{-1} by $\Delta^{-1} U_i$ (perm braid)

  & collect all Δ^{-1}'s to the left

  $\exists$ smarter ways to collect more letters into a same B_i when obviously possible]

- if $S(B_{i+1}) \subset F(B_i)$ $\forall i$ we're done! (simply the first few B_i might be Δ, the last few might be e, collect appropriately)

- otherwise, pick $j \in S(B_{i+1}) \setminus F(B_i)$, replace $B_i \leftarrow B_i \sigma_j$

  $B_{i+1} \leftarrow \sigma_j^{-1} B_{i+1}$

  (gives something with $(\deg B_1, \dots, \deg B_k)$ lexicographically larger) & repeat.

  ⇝ terminates in finite # steps

  (not too large if we do things in the right order...).

Machine presentation = a bunch of permutation tables ($\pi_i \in \mathfrak{S}_n$ corresp. to the factors).

$\exists$ efficient ways to manipulate them (and starting/finishing sets easy to read off!).

If optimize algorithm a bit (to transfer as much as possible from B_{i+1} to B_i in single step)

For a word of length ℓ in B_n, can compute the normal form in $O(\ell^2 n \log n)$.

Conjugacy problem: for the details (see Elrifai-Morton, "Algorithms for positive braids" Quarterly J. Math. Oxford, 1994)

Def: given $B \in B_n$, let $r_0 = \sup\{\inf B',\ B' \text{ conjugate to } B\}$
$s_0 = \inf\{\sup B',\ \text{———}\}$
"Super-summit set" $SSS(B) = \{B' \text{ conj. to } B \,/\, B' \in [r_0, s_0]\}$
↳ a very particular set of conjugates (necess. finite since $[r_0, s_0]$ is finite)

Not clear can achieve maximal inf and minimal sup simultaneously – it is a nontrivial result that $SSS \neq \phi$.

If we can compute $SSS(B)$, this gives a sol. to conj. problem (compare $SSS(B_1)$ & $SSS(B_2)$!).

2 steps: $\left\{\right.$ – find an elt of $SSS(B)$
– given one, find all the others.

- Start with the 2nd part. key property:

Prop: Assume $P, Q \geq \Delta^r$ are conjugate; can assume $Q = A^{-1}PA$ for some $A \in B_n^+$; let A_1 = first factor in the left-canonical form of $A = A_1 \dots A_k$ (allowing Δ ... here). Then $A_1^{-1}PA_1 \geq \Delta^r$.

this is because Δ^2 is central → mult. A by Δ^2 until $A \geq e$.

There's a similar statement about sups. (by considering P^{-1} & Q^{-1}).

Corollary: $P, Q \in [r, s]$ mutually conjugate $\Rightarrow \exists$ sequence $P = P_0, P_1, \dots, P_k = Q$ s.t. $P_i \in [r,s]\ \forall i$, and P_{i+1} = conjugate of P_i by a permutation braid.

(consider successively conjugation by the canonical factors $A_1, \dots, A_k$).
– stays in $[r,s]$ by prop^{n} –

Hence: once we have an element $P \in SSS(B)$, we know we can construct the entire set $SSS(B)$ by repeating: $\left\{\right.$ – conjugate by all perm. braids
– compute canonical form, check if we get elts of SSS
– iterate for all new elts of SSS found in previous step.

• How to find an elt of SSS ?

Def: <u>cycling</u> := given $P = \Delta^r A_1 \cdots A_k$ left-canonical form,

$$c(P) := \Delta^r A_2 \cdots A_k \tau^r(A_1), \quad \text{where } \tau = \text{conj. by } \Delta = \sigma_i \mapsto \sigma_{n-i}$$

(a particular conjugate of P, by $\tau^r(A_1)$; this expression isn't necess. its left-can. form, but little work needed to compute left-can ($O(k)$ instead of $O(k^2)$) because the part $A_2 \cdots A_k$ is already left weighted).

• Clearly, $\inf c(P) \geq r$ (in fact it's either r or $r+1$)
$\sup c(P) \leq r+k$ ($r+k$ or $r+k-1$).

Lemma: Assume $\exists Q$ conj. to P with $\inf Q > \inf P$. (i.e. $\inf(P)$ not maximal)
Then repeated cycling will produce $c^d(P)$ with $\inf c^d(P) > \inf P$.

Moreover cycling is <u>eventually periodic</u> (since stays in the finite set $[r, r+k]$) - so we can find a conjugate with maximal inf by repeatedly cycling until we hit a same conjugate twice.

In fact, $\exists$ <u>estimate</u> on # cyclings needed at most to increase $\inf(P)$ if possible: namely, need at most $\frac{n(n-1)}{2}$ cyclings (for $P \in B_n$)
(so if inf doesn't increase in $\frac{n(n-1)}{2}$ steps, we're done).
in particular we get the maximal inf in polynomial time.

• To minimize sup, similarly perform <u>decycling</u> ($\Leftrightarrow$ cycling on normal form of P^{-1})

$$P = \Delta^r A_1 \cdots A_k \rightsquigarrow d(P) = \Delta^r \tau^r(A_k) A_1 \cdots A_{k-1}$$

<u>normal form</u> (& need to recompute normal form!)

Decycling doesn't decrease inf, and eventually decreases sup if possible

$\Rightarrow$ get an element of SSS by
- cycling until inf maximal
- decycling until sup minimal!

(polynomial time). (but other part, deriving all of SSS, is expl at worst).

• <u>Further improvement</u> (Birman-Ko-Lee 1997):
replace Artin generators by <u>band generators</u>

- generators: $a_{ij} = \sigma_{j-1} \cdots \sigma_{i+1}\sigma_i\sigma_{i+1}^{-1}\cdots\sigma_{j-1}^{-1}$, $1 \le i < j \le n$

  = exch. i & j along

  = square roots of generators of P_n.

- relations:
  - $a_{ij}a_{kl} = a_{kl}a_{ij}$ if arcs $\cap = \emptyset$

    i.e. $(k-i)(k-j)(l-i)(l-j) > 0$
  - $a_{jk}a_{ij} = a_{ik}a_{jk} = a_{ij}a_{ik}$ $\forall i<j<k$

- there is a new notion of positive words wrt band generators (positive words in a_{ij}) ; Garside embedding thm holds ✓

- Garside elt is replaced by $\delta = \sigma_{n-1}\cdots\sigma_1 = a_{n-2,n-1}\cdots a_{1,2}$

  (now $\delta^n = \Delta^2$)

- Elrifai-Morton's ideas for normal forms, word & conj. problems extend with the obvious modifications

  → get a normal form $P = \delta^r A_1 \cdots A_k$, $e < A_i < \delta$, left weighted wrt a_{ij}'s.

  (elements of $[e,\delta]$ are a proper subset of permutation braids: those whose permutation = product of parallel descending cycles $(s_i \ldots s_1)$, $s_i > \cdots > s_1$, s.t. $s_1\ s_2 \ldots s_i$ mutually disjoint)

  Their # is the Catalan number $C_n = \frac{1}{n+1}\binom{2n}{n}$ ($\ll n!$). (grows "like" 4^n).

Theory remains the same, but word & conj. algorithms are a little bit faster (e.g. normal form in $O(\ell^2 n)$ instead of $O(\ell^2 n \log n)$; max # cyclings/decyclings to change inf/sup is n instead of $\frac{n(n-1)}{2}$)

Braid cryptography:

Two foundational papers (many others since):
- Anshel-Anshel-Goldfeld MRL 1999
- Ko, Lee, Cheon, … 2000: Proc. Crypto 2000, Springer LNCS 1880

attempt to develop public-key cryptography that doesn't rely on number theory.

Starting point: braids are easy to implement, but conjugacy problem is hard in general. [however, one must be very careful in how to choose random braids!]

Easiest framework: $\boxed{\text{key agreement protocol}}$ (A & B communicate on an open channel in order to establish a common secret (known only to them) used to encrypt further transmissions.
(crypto theory $\Rightarrow$ can convert this into other things such as authentication schemes, public key encryption schemes...)

Ko-Lee-Cheon-...

Setup: consider a braid group on $(\ell+r)$ strings $B_{\ell+r} \supset LB_\ell$ left ℓ strings
RB_r right r strings
elts of LB_ℓ & RB_r commute w/ each other

- Public data: $x \in B_{\ell+r}$ "sufficiently complicated" (chosen by either A or B)

- A chooses $a \in LB_\ell$ (secret) & announces $y = axa^{-1}$ (public).

  [so... need to hope that conj. search problem, i.e. finding a given x & axa^{-1}, is hard!!].

- B chooses $b \in RB_r$ (secret) & announces $z = bxb^{-1}$ (public)

- Common secret: $byb^{-1} = abx(ab)^{-1} = aza^{-1}$ (recall $ab=ba$!! LB_ℓ, RB_r commute!)

  Use this to encode all messages (in any way one wants) (many classical crypto protocols) $\to$ communicate securely from this point on

  [so... security relies on: knowing x, axa^{-1}, bxb^{-1}, can't find $abx(ab)^{-1}$! generally thought to be equivalent to CSP – need to get a or b but not well justified...]

The common secret can be computed from public data (x,y,z) + either one of a or b (but hopefully not without a and b).

(Turning this into public-key crypto: A publishes (x,y) public key, keeps a private key
To send a message m to A:
- B chooses b at random
- B sends $z = bxb^{-1}$, & the message encoded using byb^{-1}.
- A decodes using aza^{-1})

Anshel-Anshel-Goldfeld:

- Public data: 2 subgroups $S_A = \langle s_1, \dots, s_m \rangle$, $S_B = \langle t_1, \dots, t_m \rangle$
- A secretly chooses $a \in S_A$, & makes public $at_1a^{-1}, \dots, at_ma^{-1}$
  B $\quad b \in S_B \quad bs_1b^{-1}, \dots, bs_mb^{-1}$
- then A & B can both compute $[a,b]$ = shared secret

for A: $a = \prod s_{i_k}^{\pm 1} \Rightarrow aba^{-1}b^{-1} = \prod s_{i_k}^{\pm 1} \cdot \left(\prod (b s_{i_k} b^{-1})^{\pm 1}\right)^{-1}$,

public key of B

for B: $b = \prod t_{j_k}^{\pm 1} \Rightarrow aba^{-1}b^{-1} = \left(\prod (a t_{i_k} a^{-1})^{\pm 1}\right) \cdot \left(\prod t_{j_k}^{\pm 1}\right)^{-1}$

public key of A.

[again: it's generally thought that $[a,b]$ can't be computed without knowing either a or b i.e. solving CSP].