Il problema del coniugio nei gruppi di trecce

Leonardo Migliorini

1 Introduzione

Il problema del coniugio in un gruppo arbitrario G consiste nel determinare se due elementi x e y di G sono coniugati, cioè se esiste un elemento z di G tale che $z^{-1}xz = y$ (d'ora in avanti useremo anche la notazione x^z per indicare $z^{-1}xz$). Questo è uno dei principali problemi decisionali nella teoria dei gruppi, che è tuttavia indecidibile in generale. Nel caso dei gruppi di trecce il problema viene risolto da Garside in [Gar69] e tale soluzione è applicabile a una classe più ampia di gruppi, chiamati *gruppi di Garside*. La soluzione di Garside consiste nel calcolo di un sottoinsieme del gruppo in questione, detto *summit set*, che risulta essere invariante completo per coniugio. Nel corso degli anni sono stati apportati diversi miglioramenti alla soluzione proposta da Garside basati sul calcolo di sottoinsiemi del summit set, anch'essi invarianti completi. Le soluzioni proposte tuttavia non raggiungono una complessità polinomiale, sono quindi lontane da un'implementazione efficiente. Nel seguito descriviamo l'algoritmo proposto in [GGM10b], che risolve il problema calcolando un sottoinsieme del summit set detto *insieme degli sliding circuits*, e vediamo alcune famiglie di esempi in cui la cardinalità dell'insieme di interesse cresce esponenzialmente. L'algoritmo in questione risolve in problema in un generico gruppo di Garside, anche se la sperimentazione verrà fatta sui gruppi di trecce. La referenza principale per i risultati che vengono utilizzati è [GGM10b], a meno che non venga indicato diversamente.

2 Nozioni preliminari

Definiamo in generale i gruppi di Garside, di cui i gruppi di trecce sono un caso particolare, e vediamo alcune proprietà rilevanti per la soluzione del problema del coniugio.

Definizione 2.1. Un gruppo G si dice *gruppo di Garside* con *struttura di Garside* (G, P, Δ) se ammette un sottomonoide P tale che $P \cap P^{-1} = \{1\}$ e un elemento $\Delta \in G$ con le seguenti proprietà:

(G1) Il gruppo G con la relazione d'ordine $\preccurlyeq$ tale che $a \preccurlyeq b$ se e solo se $a^{-1}b \in P$ è un *reticolo*. Questo significa che per ogni $a, b \in G$ esistono unici il *massimo comune divisore* $a \wedge b$ e il *minimo comune multiplo* $a \vee b$ rispetto all'ordine $\preccurlyeq$.

(G2) L'insieme $[1, \Delta] = \{a \in G \mid 1 \preccurlyeq a \preccurlyeq \Delta\}$, i cui elementi si dicono *elmenti semplici*, genera G.

(G3) Il monoide P è invariante per coniugio per Δ.

(G4) Per ogni $x \in P \setminus \{1\}$ la quantità

$$\|x\| = \sup\{k \in \mathbb{N} \mid \exists a_1, \dots, a_k \in P \setminus \{1\} \text{ per cui } x = a_1 \dots a_k\}$$

è finita.

Il monoide P è il *monoide degli elementi positivi*, mentre Δ è l'*elemento di Garside*. Una struttura di Garside (G, P, Δ) si dice di *tipo finito* se l'insieme $[1, \Delta]$ è finito. Se un gruppo di Garside G ammette una struttura di Garside di tipo finito diciamo che G è di tipo finito.

Chiamiamo *atomi* gli elementi $x \in P$ tali che se $x = yz$ allora $y = 1$ oppure $z = 1$. Questi sono in numero finito e costituiscono un insieme di generatori per G.

Il *gruppo di trecce su n fili* è il gruppo definito dalla seguente presentazione:

$$B_n = \left\langle \sigma_1, \dots, \sigma_{n-1} \;\middle|\; \begin{array}{ll} \sigma_i\sigma_j = \sigma_j\sigma_i & \text{se } |i-j| > 1 \\ \sigma_i\sigma_j\sigma_i = \sigma_j\sigma_i\sigma_j & \text{se } |i-j| = 1 \end{array} \right\rangle$$

Nel caso di B_n il monoide degli elementi positivi è dato dal monoide B_n^+ generato dalle potenze positive di $\sigma_1, \dots, \sigma_{n-1}$, mentre l'elemento di Garside è

$$\Delta = \sigma_1(\sigma_2\sigma_1)(\sigma_3\sigma_2\sigma_1)\dots(\sigma_{n-1}\dots\sigma_1)$$

Gli elementi semplici $[1, \Delta]$ sono $n!$, quindi questa struttura di Garside è di tipo finito. Gli elementi $\sigma_1, \dots, \sigma_{n-1}$ sono gli atomi di B_n.

Consideriamo d'ora in poi un gruppo di Garside G con struttura di Garside (G, P, Δ) di tipo finito, diamo alcune definizioni preliminari.

Possiamo definire su G un ordinamento $\succcurlyeq$ in modo simmetrico rispetto a $\preccurlyeq$: scriviamo che $a \succcurlyeq b$ se e solo se $ab^{-1} \in P$. Questo ha proprietà analoghe a quelle di $\preccurlyeq$, in particolare ogni coppia di elementi $a, b \in G$ ammette un unico massimo comune divisore $a \wedge^{\gamma} b$ e un unico minimo comune multiplo $a \vee^{\gamma} b$ rispetto a $\succcurlyeq$.

Definizione 2.2. Sia $x \in G$, diciamo che una scrittura del tipo $x = \Delta^p x_1 \dots x_r$ è una *forma normale sinistra* di x se $x_i \in [1, \Delta] \setminus \{1, \Delta\}$ per ogni $i = 1, \dots, r$ e se $x_i x_{i+1} \wedge \Delta = x_i$ per ogni $i = 1, \dots, r-1$. Analogamente diciamo che una scrittura del tipo $x = y_1 \dots y_s \Delta^q$ è una *forma normale destra* di x se $y_i \in [1, \Delta] \setminus \{1, \Delta\}$ per ogni $i = 1, \dots, s$ e se $y_{i+1} y_i \wedge^{\gamma} \Delta = y_i$ per ogni $i = 1, \dots, s-1$.

È un fatto noto che nei gruppi di Garside ogni elemento ammette un'unica forma normale (destra o sinistra). Vale inoltre che $p = q$ e $r = s$ con le notazioni della definizione. Se $x \in G$ ha forma normale sinistra $x = \Delta^p x_1 \dots x_r$ poniamo $\inf(x) = p$, $\ell(x) = r$ e $\sup(x) = p + r$. La quantità $\ell(x)$ si dice *lunghezza canonica* di x. In [EM94] viene mostrato che $\inf(x)$ e $\sup(x)$ sono rispettivamente il massimo e il minimo intero per cui vale $\Delta^{\inf(x)} \preccurlyeq x \preccurlyeq \Delta^{\sup(x)}$.

Definizione 2.3. Sia $x \in G$, consideriamo l'elemento

$$\mathfrak{p}(x) = (x\Delta^{-\inf(x)}) \wedge (x^{-1}\Delta^{\sup(x)}) \wedge \Delta$$

Chiamiamo *cyclic sliding* di x l'elemento

$$\mathfrak{s}(x) = x^{\mathfrak{p}(x)} = \mathfrak{p}(x)^{-1}x\mathfrak{p}(x)$$

In modo analogo possiamo definire

$$\mathfrak{p}^{\curvearrowleft}(x) = (\Delta^{-\inf(x)}x) \wedge^{\curvearrowleft} (\Delta^{\sup(x)}x^{-1}) \wedge^{\curvearrowleft} \Delta, \quad \mathfrak{s}^{\curvearrowleft}(x) = x^{\mathfrak{p}^{\curvearrowleft}(x)^{-1}} = \mathfrak{p}^{\curvearrowleft}(x)x\mathfrak{p}^{\curvearrowleft}(x)^{-1}$$

Definizione 2.4. Diciamo che un elemento $y \in G$ appartiene a uno *sliding circuit* se è fissato da una potenza del cyclic sliding, cioè se esiste $m \geq 1$ per cui $\mathfrak{s}^m(y) = y$. Dato $x \in G$ chiamiamo *insieme degli sliding circuits* l'insieme $SC(x)$ costituito dai coniugati di x che appartengono a uno sliding circuit.

L'insieme degli sliding circuits SC è un sottoinsieme del summit set che viene definito in [Gar69] ed è anch'esso un invariante completo per coniugio. Quello che viene mostrato in [GGM10b] è che per ogni $x \in G$ l'insieme $SC(x)$ è un sottoinsieme del *super summit set* $SSS(x)$.

Definizione 2.5. Sia $x \in G$, indichiamo con x^G la classe di coniugio di x in G. Consideriamo le quantità $\inf_s(x) = \max\{\inf(y) \mid y \in x^G\}$ e $\sup_s(x) = \min\{\sup(y) \mid y \in x^G\}$. Chiamiamo *super summit set* di x l'insieme

$$SSS(x) = \{y \in x^G \mid \sup_s(y) = \sup(y), \inf_s(y) = \inf(y)\}$$

Le proprietà principali del super summit set, tra cui il fatto che è invariante completo per coniugio e che è un sottoinsieme del summit set, vengono mostrate in [EM94]. Inoltre vale l'inclusione $SC(x) \subseteq SSS(x)$ ([GGM10a, Proposizione 3.13]). Sempre in [GGM10a] viene mostrato che $SC(x)$ è invariante completo per coniugio di x, in particolare $x, y \in G$ sono coniugati se e solo se $SC(x) = SC(y)$. In caso contrario $SC(x) \cap SC(y) = \emptyset$. Per risolvere il problema del coniugio è quindi sufficiente controllare se gli insiemi degli sliding circuits di x e y si intersecano.

È utile rappresentare l'insieme degli sliding circuits come un grafo orientato $SCG(x)$, il cui insieme di nodi è costituito dagli elementi di $SC(x)$. Gli archi corrispondono agli elementi che collegano due nodi via coniugio.

Definizione 2.6. Sia $x \in G$, il *grafo degli sliding circuits* $SCG(x)$ di x è il grafo orientato i cui nodi sono gli elementi di $SC(x)$ e i cui archi sono definiti dalla seguente proprietà: c'è un arco orientato da $u \in SC(x)$ a $v \in SC(x)$ associato a un elemento $s \in P \setminus \{1\}$ se e solo se:

(i) $u^s = v$.

(ii) L'elemento s è *indecomponibile*, cioè $s \neq 1$ e non esiste un elemento t tale che $1 \prec t \prec s$ e $u^t \in SC(x)$.

Nella soluzione proposta sarà necessario calcolare gli archi uscenti dai nodi di $SCG(x)$. I seguenti risultati (rispettivamente [GGM10b, Corollario 2.8] e [GGM10b, Corollario 3.0]) saranno utili a tale scopo.

Proposizione 2.7. *Sia $v \in G$. Esiste un unico elemento positivo $\rho(v) \in P$ tale che $v^{\rho(v)} \in SSS(v)$ e $\rho(v) \preccurlyeq \alpha$ per ogni $\alpha \in P$ tale che $v^\alpha \in SSS(v)$.*

Proposizione 2.8. *Sia $v \in G$. Esiste un unico elemento positivo $c(v) \in P$ tale che $v^{c(v)} \in SC(v)$ e $c(V) \preccurlyeq \alpha$ per ogni $\alpha \in P$ tale che $v^\alpha \in SC(v)$.*

Viene mostrato in [GGM10b, Corollario 3.3] che gli archi del grafo sono associati a elementi semplici e che il numero di archi uscente da ogni nodo è superiormente limitato dal numero di atomi di G.

Definiamo adesso un'operazione che useremo per calcolare gli archi del grafo $SCG(x)$.

Definizione 2.9. Siano $x, \alpha \in G$, chiamiamo *trasporto* di α in x tramite cyclic sliding l'elemento $\alpha^{(1)} = \mathfrak{p}(x)^{-1}\alpha\mathfrak{p}(x^\alpha)$. Definiamo iterativamente $\alpha^{(0)} = \alpha$ e $\alpha^{(i)} = (\alpha^{(i-1)})^{(1)}$ per $i > 0$, che è il trasporto di $\alpha^{(i-1)}$ in $\mathfrak{s}^{i-1}(x)$.

L'operazione di trasporto permette di calcolare "in modo coerente" un elemento che coniuga $\mathfrak{s}^i(x)$ in $\mathfrak{s}^i(x^\alpha)$ a partire da α. La coerenza di tale costruzione è data dal fatto che commuta il seguente diagramma (con $u \xrightarrow{s} v$ intendiamo $u^s = v$):

$$\begin{array}{ccccccccc}
x & \xrightarrow{\mathfrak{p}(x)} & \mathfrak{s}(x) & \xrightarrow{\mathfrak{p}(\mathfrak{s}(x))} & \mathfrak{s}^2(x) & \longrightarrow \dots \longrightarrow & \mathfrak{s}^{k-1}(x) & \xrightarrow{\mathfrak{p}(\mathfrak{s}^{k-1}(x))} & \mathfrak{s}^k(x) \\
{\scriptstyle \alpha^{(0)}=\alpha}\downarrow & & \downarrow{\scriptstyle \alpha^{(1)}} & & \downarrow{\scriptstyle \alpha^{(2)}} & & \downarrow{\scriptstyle \alpha^{(k-1)}} & & \downarrow{\scriptstyle \alpha^{(k)}} \\
x^\alpha & \xrightarrow[\mathfrak{p}(x^\alpha)]{} & \mathfrak{s}(x^\alpha) & \xrightarrow[\mathfrak{p}(\mathfrak{s}(x^\alpha))]{} & \mathfrak{s}^2(x^\alpha) & \longrightarrow \dots \longrightarrow & \mathfrak{s}^{k-1}(x^\alpha) & \xrightarrow[\mathfrak{p}(\mathfrak{s}^{k-1}(x^\alpha))]{} & \mathfrak{s}^k(x^\alpha)
\end{array}$$

Vale che il trasporto di un elemento semplice è ancora un elemento semplice ([GGM10b, Proposizione 2.4(5)]) e che, sotto opportune ipotesi, il trasporto lungo uno sliding circuit mantiene l'ordinamento $\preccurlyeq$. In particolare se $x, \alpha, \beta \in G$ sono tali che $x, x^\alpha, x^\beta \in SSS(x)$ e $\alpha \preccurlyeq \beta$ allora $\alpha^{(1)} \preccurlyeq \beta^{(1)}$ ([GGM10b, Lemma 2.4(4)]). Per induzione vale in generale $\alpha^{(k)} \preccurlyeq \beta^{(k)}$. Inoltre tale operazione permette di determinare se alcuni coniugati degli elementi di $SC(x)$ appartengono a $SC(x)$ ([GGM10b, Lemma 2.5]).

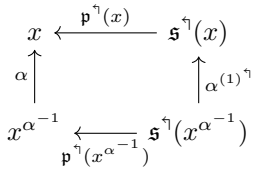
Proposizione 2.10. *Siano $x \in G$, $y \in SC(x)$ e $s \in G$ tale che $y^s \in SSS(x)$. Sia $N > 0$ un intero tale che $\mathfrak{s}^N(y) = y$. Valgono i seguenti fatti:*

(i) *Esistono interi $i_2 > i_1 \geq 0$ tali che $s^{(i_1 N)} = s^{(i_2 N)}$.*

(ii) *$y^s \in SC(x)$ se e solo se esiste un intero k tale che $s^{(kN)} = s$.*

Possiamo costruire un trasporto in modo analogo usando l'ordinamento $\succcurlyeq$. Definiamo $\alpha^{(1)^\urcorner} = \mathfrak{p}^\urcorner(x)x\mathfrak{p}^\urcorner(x)^{-1}$, in particolare $\alpha^{(1)^\urcorner}$ fa commutare il

diagramma



e $\alpha^{(k)^\gamma}$ iterativamente, come nel caso già visto.

Definizione 2.11. Siano $x \in G, z \in SC(x), y = \mathfrak{s}(z)$ e $s \in P$. Segue da [GGM10b, Proposizione 2.4(6), Proposizione 2.6] che esiste un unico elemento $\preccurlyeq$-minimale $s_{(1)} \in P$ tale che $z^{s_{(1)}} \in SSS(x)$ e $s \preccurlyeq (s_{(1)})^{(1)}$, dove per $(s_{(1)})^{(1)}$ intendiamo il trasporto di $s_{(1)}$ in z. L'elemento $s_{(1)}$ si dice *pullback* di s in y. Per $k > 1$ definiamo ricorsivamente il k-pullback di s in y come $s_{(k)} = (s_{(k-1)})_{(1)}$, intendendo il pullback di $s_{(k-1)}$ nell'unico elemento w nello sliding circuit di y tale che $\mathfrak{s}^{k-1}(w) = y$. Poniamo $s_{(0)} = s$.

Siano $z \in SC(x)$ e $y = \mathfrak{s}^k(z)$ per un intero positivo k e $s \in P$. Per [GGM10b, Lemma 3.15] il pullback $s_{(k)}$ in y è l'unico elemento positivo $\preccurlyeq$-minimale tale che $s \preccurlyeq (s_{(k)})^{(k)}$ e $z^{s_{(k)}} \in SSS(x)$. Inoltre se $t \in G$ è tale che $1 \preccurlyeq s \preccurlyeq t$ allora i loro pullback in y soddisfano $s_{(k)} \preccurlyeq t_{(k)}$ ([GGM10b, Lemma 3.16]). Dal [GGM10b, Lemma 3.17] segue anche che il pullback di un elemento semplice è ancora un elemento semplice.

3 Descrizione dell'algoritmo

L'algoritmo proposto decide se due elementi x e y sono coniugati e in caso affermativo calcola un elemento z tale che $x^z = y$. Viene mostrato in [GGM10b, Sezione 4.1] che è sufficiente conoscere la struttura di Garside di G per risolvere il problema. In particolare supponiamo di avere a disposizione:

(i) L'elemento di Garside Δ.

(ii) Una lista $\mathcal{A} = \{a_1, \ldots, a_\lambda\}$ contenente gli atomi di G.

(iii) Una funzione che, dato $a \in \mathcal{A}$ e $s \in [1, \Delta]$, determina se $a \preccurlyeq s$ e in tal caso calcola $a^{-1}s$.

(iv) Una funzione che, dato $a \in \mathcal{A}$ e $s \in [1, \Delta]$, determina se $a \succcurlyeq s$ e in tal caso calcola sa^{-1}.

L'algoritmo è suddiviso in tre parti, di cui diamo separatamente lo pseudo-codice.

Algoritmo 1:
Calcolo di un elemento di $SC(x)$.

Input: $x \in G$.
Output: $\tilde{x} \in SC(x)$ e $c \in G$ tale che $x^c = \tilde{x}$.

(1) $\tilde{x} = x,\ c = 1, \mathcal{T} = \emptyset$;
(2) **while** $\tilde{x} \notin \mathcal{T}$ **do:** $\mathcal{T} = \mathcal{T} \cup \{\tilde{x}\},\ c = c \cdot \mathfrak{p}(\tilde{x}),\ \tilde{x} = \mathfrak{s}(\tilde{x})$;
(3) $y = \mathfrak{s}(\tilde{x}),\ d = \mathfrak{p}(\tilde{x})$;
(4) **while** $y \neq \tilde{x}$ **do:** $d = d \cdot \mathfrak{p}(y),\ y = \mathfrak{s}(y)$;
(5) **return** $\tilde{x},\ c = c \cdot d^{-1}$;

Algoritmo 2:
Calcolo degli archi di $SCG(x)$ uscenti da un nodo dato.

Input: $v \in SC(x)$.
Output: L'insieme $\mathcal{A}_v$ degli archi del grafo $SCG(x)$ uscenti da v.

(1) Calcolo del minimo intero $N > 0$ tale che $\mathfrak{s}^N(v) = v$;
(2) $L = \{a_1, \ldots, a_\lambda\}$ lista degli atomi di $G,\ \mathcal{A}_v = \emptyset,\ Atomi = \emptyset$;
(3) **for** $t = 1, \ldots, \lambda$ **do:**
(a) $s = a_t$;
(b) **while** $\ell(v^s) > \ell(v)$ **do:** $s = s \cdot (1 \vee (v^s)^{-1}\Delta^{\inf(v)} \vee v^s\Delta^{-\sup(v)})$;
(c) **if** $a_t \preccurlyeq \mathfrak{p}(v)$ **then:** Calcolo degli N-pullbacks $s,\ s_{(N)},\ s_{(2N)}, \ldots$ fino alla prima ripetizione $s_{(rN)},\ s = s_{(rN)}$;
(d) Calcolo degli N-trasporti $s,\ s^{(N)},\ s^{(2N)}, \ldots$ fino alla prima ripetizione $s^{(jN)}$, sia $i < j$ tale che $s^{(iN)} = s^{(jN)}$;
(e) **if** $a_t \preccurlyeq s^{(mN)}$ per un m tale che $i \leq m < j$ **then do:**
(i) **if** $a_k \not\preccurlyeq s^{(mN)}$ per $k = 1, \ldots, \lambda$ tale che $a_k \in Atomi$ oppure $k > t$ **then do:** $\mathcal{A}_v = \mathcal{A}_v \cup \{s^{(mN)}\},\ Atomi = Atomi \cup \{a_t\}$;
(4) **return** $\mathcal{A}_v$;

Algoritmo 3:
Soluzione del problema del coniugio in G.

Input: $x, y \in G$.
Output: Sì/No (a seconda che x, y siano coniugati) e $c \in G$ tale che $x^c = y$.

(1) Calcolo di $\tilde{x} \in SC(x)$ e $\tilde{y} \in SC(y)$ usando l'**Algoritmo 1**, insieme agli elementi c_1 e c_2 tali che $x^{c_1} = \tilde{x}$ e $y^{c_2} = \tilde{y}$;
(2) $\mathcal{V} = \{\tilde{x}\},\ \mathcal{V}' = \{\tilde{x}\},\ c_{\tilde{x}} = 1$;
(3) **while** $\mathcal{V}' \neq \emptyset$ **do:**
  (a) **choose** $v \in \mathcal{V}'$;
  (b) Calcolo di $\mathcal{A}_v$ usando l'**Algoritmo 2**;
  (c) **for all** $s \in \mathcal{A}_v$ **do:**
    (i) **if** $v^s = \tilde{y}$ **then** $c_{\tilde{y}} = c_v \cdot s$, **return** $c = c_1 \cdot c_{\tilde{y}} \cdot c_2^{-1}$;
    (ii) **if** $v^s \notin \mathcal{V}$ **then** $c_{v^s} = c_v \cdot s,\ \mathcal{V} = \mathcal{V} \cup \{v^s\},\ \mathcal{V}' = \mathcal{V}' \cup \{v^s\}$;
  (d) **remove** v **from** $\mathcal{V}'$;
(4) **return** "x e y non sono coniugati.";

3.1 Algoritmo 1

Dato $x \in G$ l'Algoritmo 1 determina due elementi $\tilde{x} \in SC(x)$ e $c \in G$ tali che $x^c = \tilde{x}$ applicando iterativamente il cyclic sliding. Questa procedura termina, infatti viene dimostrato in [GGM10a, Corollario 2.2] che esistono degli interi $0 \leq i < j$ tali che $\mathfrak{s}^i(x) = \mathfrak{s}^j(x)$. In particolare $\mathfrak{s}^i(x)$ è un elemento di $SC(x)$ dato che è fissato da $\mathfrak{s}^{j-i}$. L'algoritmo calcola quindi l'elemento $\mathfrak{s}^i(x) \in SC(x)$ con i minimale. Questo viene fatto memorizzando nell'insieme $\mathcal{T}$ gli elementi $\mathfrak{s}^m(x)$ per $m \geq 0$ e $\tilde{x}$ viene scelto come il primo elemento che viene aggiunto per la seconda volta a $\mathcal{T}$. Questo calcolo, insieme a quello di c, viene eseguito nel passi 1, 2 dell'algoritmo.

Se la prima ripetizione in $\mathcal{T}$ avviene alla k-esima iterazione allora esiste $0 \leq h < k$ tale che $\mathfrak{s}^h(x) = \mathfrak{s}^k(x)$, possiamo quindi semplificare l'elemento $c = \mathfrak{p}(x) \cdot \mathfrak{p}(\mathfrak{s}(x)) \cdot \ldots \cdot \mathfrak{p}(\mathfrak{s}^k(x))$ che trasporta x in $\tilde{x}$. Infatti dal momento che $\mathfrak{s}^h(x) = \mathfrak{s}^k(x) = \tilde{x}$ è sufficiente considerare l'elemento $\mathfrak{p}(x) \cdot \mathfrak{p}(\mathfrak{s}(x)) \cdot \ldots \cdot \mathfrak{p}(\mathfrak{s}^h(x))$ ottenuto eliminando gli ultimi $k - h + 1$ fattori nella scrittura di c. I passi 3, 4, 5 dell'algoritmo si occupano di questo calcolo.

3.2 Algoritmo 2

L'Algoritmo 2 è il più complesso tra quelli presentati. Dato un nodo $v \in SC(x)$, l'algoritmo calcola l'insieme $\mathcal{A}_v$ degli archi uscenti da v determinando degli elementi indecomponibili che coniugano v in altri elementi di $SC(x)$. Degli elementi importanti per il calcolo degli archi sono determinati dai seguenti risultati (rispettivamente [GGM10b, Corollario 3.1] e [GGM10b, Corollario 3.2]).

Proposizione 3.1. *Siano $x \in G$, $v \in SSS(x)$, $s \in G$ e $\rho_s(v) = s \cdot \rho(v)$. Allora $\rho_s(v)$ è l'unico elemento $\preccurlyeq$-minimale per cui vale $s \preccurlyeq \rho_s(v)$ e $v^{\rho_s(v)} \in SSS(x)$. Inoltre $\rho_s(v) \preccurlyeq \Delta^{\sup(s)}$.*

Proposizione 3.2. *Siano $x \in G$, $v \in SC(x)$, $s \in G$ e $c_s(v) = s \cdot c(v)$. Allora $c_s(v)$ è l'unico elemento $\preccurlyeq$-minimale per cui vale $s \preccurlyeq c_s(v)$ e $v^{c_s(v)} \in SC(x)$. Inoltre $c_s(v) \preccurlyeq \Delta^{\sup(s)}$.*

Notiamo che $s \preccurlyeq \rho_s \preccurlyeq c_s$ dato che vale l'inclusione $SC(x) \subseteq SSS(x)$. È sufficiente considerare l'insieme degli elementi semplici $\{c_a(v) \mid a \text{ atomo di } G\}$ per determinare gli archi uscenti da v. Siano $a_t \in G$ un atomo e $N > 0$ il minimo intero per cui $\mathfrak{s}^N(v) = v$, poniamo $\rho_{a_t}(v) = \rho_{a_t}$ e $c_{a_t}(v) = c_{a_t}$. Per definizione $v^{c_{a_t}} \in SC(x)$, quindi c_{a_t} è un punto fisso per un'opportuna potenza del trasporto lungo lo sliding circuit contenente v. Per calcolare c_{a_t} cerchiamo un opportuno elemento p_{a_t} tale che $a_t \preccurlyeq p_{a_t} \preccurlyeq c_{a_t}$ che venga mandato in c_{a_t} da una potenza sufficientemente grande del trasporto. Per mantenere l'ordinamento serve che $v^{p_{a_t}} \in SSS(x)$ e che $a_t \preccurlyeq p_{a_t}^{(kN)}$ per k sufficientemente grande.

Un primo candidato per p_{a_t} è ρ_{a_t}, l'unico elemento $\preccurlyeq$-minimale tale che $a_t \preccurlyeq \rho_{a_t}$ e $v^{\rho_{a_t}} \in SSS(x)$. In [GGM10b, Proposizione 3.4] e in [GGM10b, Corollario 3.5] viene mostrato che i passi 3(a) e 3(b) calcolano $\rho_{a_t}(v)$ e il ciclo viene eseguito al più $\|\Delta\|$ volte.

Dai lemmi [GGM10b, Lemma 3.10] e [GGM10b, Corollario 3.13] segue che se $a_t \not\preccurlyeq \mathfrak{p}(v)$ allora c_{a_t} non è indecomponibile (e quindi non può rappresentare un arco uscente da v) oppure esiste un intero k tale che $c_{a_t} = (\rho_{a_t})^{(kN)}$. In questo caso è sufficiente calcolare i trasporti iterati di ρ_{a_t} per determinare se c_{a_t} rappresenta un arco o no.

Se invece $a_t \preccurlyeq \mathfrak{p}(v)$ consideriamo gli N-pullback di ρ_{a_t}. Dalla [GGM10b, Proposizione 3.18] si ha che esistono degli interi $0 \leq i < j$ tali che $(\rho_{a_t})_{(iN)} = (\rho_{a_t})_{(jN)}$ e che esiste un intero k tale che $c_{\rho_{a_t}} = ((\rho_{a_t})_{(iN)})^{(kN)}$. D'altra parte valgono le relazioni

$$\begin{cases} a_t \preccurlyeq \rho_{a_t} \preccurlyeq c_{a_t} \\ v^{c_{a_t}} \in SC(x) \end{cases} \qquad \begin{cases} \rho_{a_t} \preccurlyeq c_{\rho_{a_t}} \\ v^{c_{\rho_{a_t}}} \in SC(x) \end{cases}$$

e quindi $c_{a_t} = c_{\rho_{a_t}}$ per unicità. In questo caso possiamo quindi calcolare c_{a_t} considerando prima i pullbacks di ρ_{a_t}, iterando il trasporto in seguito. I passi 3(c), se eseguito, e 3(d) calcolano quindi l'elemento c_{a_t}. Sia $s = \rho_{a_t}$ oppure $s = (\rho_{a_t})_{(iN)}$, per il [GGM10b, Lemma 3.10] se un trasporto $s^{(mN)}$ è tale che $a_t \preccurlyeq s^{(kN)}$ allora $s^{(kN)} = c_{a_t}$. Altrimenti c_{a_t} non è indecomponibile per [GGM10b, Lemma 3.10], [GGM10b, Corollario 3.13] e [GGM10b, Proposizione 3.18].

Anche se vale $s^{(mN)} = c_{a_t}$, non è detto che c_{a_t} sia indecomponibile: questo viene controllato al passo 3(e)i. Alla fine dell'algoritmo l'insieme *Atomi* conterrà gli atomi a_k tali che c_{a_k} è indecomponibile e $k = \max\{i \mid a_i \preccurlyeq c_{a_k}\}$. Se t non è il massimo indice per cui vale $a_t \preccurlyeq c_{a_t}$ allora c_{a_t} viene scartato, dal momento che se è indecomponibile verrà considerato nelle iterazioni successive. Se invece $t = \max\{i \mid a_i \preccurlyeq c_{a_t}\}$ ma c_{a_t} è indecomponibile allora esiste un

$l < t$ tale che $c_{a_l} \preccurlyeq c_{a_t}$ e c_{a_l} è indecomponibile. In questo caso a_l è stato considerato in un'iterazione precedente, cioè $a_l \in Atomi$. Quindi se $a_k \not\preccurlyeq c_{a_t}$ per ogni $a_k \in Atomi$ e per ogni $k > t$, necessariamente c_{a_t} è indecomponibile e aggiungiamo a_t all'insieme $Atomi$.

3.3 Algoritmo 3

Siano $x, y \in G$ e $\tilde{x}, c_1, \tilde{y}, c_2 \in G$ calcolati dall'Algoritmo 1 tali che $x^{c_1} = \tilde{x} \in SC(x)$ e $y^{c_2} = \tilde{y} \in SC(y)$. Sappiamo che x e y sono coniugati se e solo se $SC(x) = SC(y)$, e in caso contrario $SC(x) \cap SC(y) = \emptyset$. È quindi sufficiente vedere se $\tilde{y}$ è un elemento di $SC(x)$ o no. L'Algoritmo 3 calcola un albero di copertura del grafo $SCG(x)$ a partire da $\tilde{x}$, in particolare un elemento c_v tale che $\tilde{x}^{c_v} = v$ per ogni $v \in SC(x)$. In questo modo se x e y sono coniugati sappiamo determinare un elemento c tale che $x^c = y$.

Il calcolo dell'albero di copertura viene eseguito come segue. Definiamo inizialmente due insiemi $\mathcal{V} = \mathcal{V}' = \{\tilde{x}\}$ contenenti rispettivamente gli elementi di $SC(x)$ che sono stati calcolati e gli elementi di $\mathcal{V}$ che non sono ancora stati utilizzati per il calcolo degli archi del grafo. Poniamo anche $c_{\tilde{x}} = 1$: in generale per $v \in SC(x)$ calcoliamo l'elemento c_v tale che $\tilde{x}^{c_v} = v$.

Scelto $v \in \mathcal{V}'$ calcoliamo degli elementi di G che rappresentano gli archi di $SCG(x)$ uscenti da v con l'Algoritmo 2. Per ognuno di questi elementi $s \in G$ calcoliamo v^s. Se $v^s = \tilde{y}$ allora $\tilde{y} = \tilde{x}^{c_s}$, quindi $c = c_1 \cdot c_v \cdot c_2^{-1}$ è tale che $x^c = y$. Se invece $v^s \neq \tilde{y}$ controlliamo se v^s è un elemento di $\mathcal{V}$: in caso contrario lo aggiungiamo a $\mathcal{V}$ e $\mathcal{V}'$, calcoliamo $c_{v^s} = c_v \cdot s$. Una volta esaminati tutti gli archi rimuoviamo v da $\mathcal{V}'$.

Notiamo che l'algoritmo termina in quanto a ogni passo viene aggiunto al più un elemento a $\mathcal{V}'$, mentre ne viene rimosso sempre esattamente uno. L'Algoritmo 3 risolve quindi il problema del coniugio.

4 Complessità computazionale

In [GGM10b, §4.1.3] vengono mostrati degli algoritmi efficienti per il calcolo delle operazioni principali all'interno di un gruppo di Garside. Supponiamo che siano noti gli atomi $\{a_1, \ldots, a_\lambda\}$ di G e l'elemento di Garside Δ e che il costo computazionale per eseguire la seguente operazione sia $O(C)$ (dove C è una costante):

- dato a un atomo e s un elemento semplice, verificare se $a \preccurlyeq s$ (rispettivamente $a \succcurlyeq s$) e nel caso calcolare $a^{-1}s$ (rispettivamente sa^{-1}).

Vale il seguente risultato ([GGM10b, Teorema 4.4]).

Teorema 4.1. *Sia G un gruppo di Garside di tipo finito con atomi $\{a_1, \ldots, a_\lambda\}$ ed elemento di Garside Δ. Siano a un atomo, s, t elementi semplici, x, y e α elementi di G scritti come prodotto di al più k elementi semplici o inversi di elementi semplici.*

(i) *Il calcolo di queste operazioni ha costo computazionale* $O(C\lambda)$*:*

- *Verificare se* $s = 1$.

(ii) *Il calcolo di queste operazioni ha costo computazionale* $O(C\lambda\|\Delta\|)$*:*

- *Verificare se* $s = t$.
- *Calcolare* $\Delta^{-1}s\Delta$ *oppure* $\Delta s\Delta^{-1}$.
- *Verificare se* as *è semplice e nel caso calcolarlo.*
- *Verificare se* sa *è semplice e nel caso calcolarlo.*
- *Calcolare* $s \wedge t$, $s \wedge^{\curvearrowleft} t$, $s \vee t$, $s \vee^{\curvearrowleft} t$.

(iii) *Il calcolo di queste operazioni ha costo computazionale* $O(C\lambda k\|\Delta\|)$*:*

- *Verificare se* $x = y$ *se sono note le forme normali (destra o sinistra) di* x *e* y *sono note.*
- *Calcolare la forma normale sinistra (rispettivamente destra) di* $xs, sx, xs^{-1}.s^{-1}x, x^s.x^{s^{-1}}$ *se è nota la forma normale sinistra (rispettivamente destra) di* x.
- *Calcolare* $\mathfrak{p}(x), \mathfrak{s}(x)$ *(rispettivamente* $\mathfrak{p}^{\curvearrowleft}(x), \mathfrak{s}^{\curvearrowleft}(x)$*) se è nota la forma normale sinistra (rispettivamente destra) di* x.
- *Calcolare il trasporto* $s^{(1)}$ *di* s *in* x *(rispettivamente* $s^{(1)^{\curvearrowleft}}$*) se è nota la forma normale sinistra (rispettivamente destra) di* x.
- *Calcolare il pullback* $s_{(1)}$ *di* s *in* x *se esiste e se è nota la forma normale sinistra dell'elemento* $z \in SC(x)$ *tale che* $\mathfrak{s}(z) = x$.

(iv) *Il calcolo di queste operazioni ha costo computazionale* $O(C\lambda k^2\|\Delta\|)$*:*

- *Calcolare la forma normale destra o sinistra di* x.
- *Calcolare* $c \wedge y$, $x \vee y$, $x \wedge^{\curvearrowleft} y$, $x \vee^{\curvearrowleft} y$.
- *Calcolare il trasporto* $\alpha^{(1)}$ *di* α *in* x *oppure* $\alpha^{(1)^{\curvearrowleft}}$.
- *Calcolare il pullback* $\alpha_{(1)}$ *di* α *in* x *se esiste e se è noto l'elemento* $z \in SC(x)$ *tale che* $\mathfrak{s}(z) = x$.

Sia x un elemento di G scritto come prodotto di al più k elementi semplici o inversi di elementi semplici. Definiamo delle quantità che saranno utili per limitare superiormente la complessità computazionale degli Algoritmi 1, 2, 3.

- Sia T un intero per cui esistono $0 \leq i < j \leq T$ tali che $\mathfrak{s}^i(x) = \mathfrak{s}^j(x)$.
- Sia M un intero tale che per ogni $z \in SC(x)$ esiste $0 < N \leq M$ con $\mathfrak{s}^N(z) = z$.
- Sia R un intero tale che per ogni $z \in SC(x)$ e per ogni elemento semplice s con $z^s \in SSS(x)$ esistono $0 \leq i < j \leq R$ per cui vale $s^{(iN)} = s^{(jN)}$, dove $\mathfrak{s}^N(z) = z$ ($s^{(m)}$ indica il trasporto di s in z).

Osservazione 4.2. Gli interi T, M, R esistono e possiamo darne delle limitazioni superiori. Infatti per [GGM10b, Corollario 2.2] l'iterazione del cyclic sliding è definitivamente ciclica, il che assicura l'esistenza di T. Per la [GGM10b, Proposizione 2.3] vale che $\mathfrak{s}^m(x) \in SSS(x)$ per ogni $m \geq k\|\Delta\|$. Dal momento che $|SC(x)| \leq (|[1,\Delta]|)^k$ possiamo scegliere $T \leq k\|\Delta\| + (|[1,\Delta]|)^k$. Inoltre, dato che $SC(x) \subseteq SSS(x)$ ed è finito, M esiste e possiamo scegliere $M \leq |SC(x)|$, in particolare $M \leq (|[1,\Delta]|)^k$. Dato che il trasporto di un elemento semplice è ancora semplice e che G è di tipo finito R esiste e possiamo scegliere $R \leq |[1,\Delta]|$.

In [GGM10b, Proposizione 4.9], [GGM10b, Proposizione 4.10] e [GGM10b, Teorema 4.11] vengono mostrati i seguenti risultati, che danno il costo computazionale degli Algoritmi 1, 2, 3.

Proposizione 4.3. *Sia G un gruppo di Garside di tipo finito con λ atomi ed elemento di Garside Δ e sia $x \in G$ scritto come prodotto di al più k elementi semplici o inversi di elementi semplici. L'Algoritmo 1 ha costo computazionale $O(C\lambda k(k+T)\|\Delta\|)$.*

Proposizione 4.4. *Sia G un gruppo di Garside di tipo finito con λ atomi ed elemento di Garside Δ. Siano $x \in G$ e $v \in SC(x)$ scritti come prodotto di al più k elementi semplici o inversi di elementi semplici. Se le forme normali destre e sinistre di x e v sono note allora l'Algoritmo 2 ha costo computazionale $O(C\lambda^2 k\|\Delta\|(\|\Delta\| + RM))$.*

Teorema 4.5. *Sia G un gruppo di Garside di tipo finito con λ atomi ed elemento di Garside Δ e siano $x, y \in G$ scritti come prodotto di al più k elementi semplici o inversi di elementi semplici. Siano T, M, R massimi rispetto alle limitazioni superiori date sopra, allora l'Algoritmo 3 ha costo computazionale $O(C\lambda k\|\Delta\|(k + T + \lambda|SC(x)|(\|\Delta\| + RM)))$.*

I limiti superiori per T e M dati nell'Osservazione 4.2 sono esponenziali in k e attualmente non sono note delle stime polinomiali. Per quanto riguarda la famiglia dei gruppi di trecce B_n si ha che la complessità dell'algoritmo è esponenziale anche in n, in quanto $|[1,\Delta]| = n!$. Per quanto riguarda l'insieme degli sliding circuits l'unica stima nota che vale in generale è $|SC(x)| \leq |SSS(x)| \leq |[1,\Delta]^k|$, che è esponenziale in k e in n per B_n. Vediamo che non è possibile migliorare qualitativamente questa stima nel caso del gruppo di trecce, nel senso che esistono degli elementi $x_n \in B_n$ per cui $|SC(x_n)|$ è esponenziale in n.

5 Risultati sperimentali

Sono stati eseguiti alcuni test dell'algoritmo proposto, implementato nel linguaggio GAP3, per verificare che la stima sulla dimensione di SC non può essere migliorata in generale. In particolare verifichiamo che per una famiglia di elementi $x_n \in B_n$ la cardinalità di $SC(x)$ è esponenziale in n, come viene mostrato in [GGM10a, Proposizione 9] e in [GM11, Proposizione 5.5].

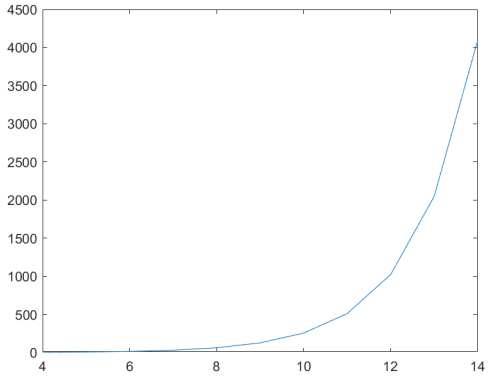
Consideriamo la presentazione standard del gruppo di trecce

$$B_n = \left\langle \sigma_1, \dots, \sigma_{n-1} \;\middle|\; \begin{array}{ll} \sigma_i\sigma_j = \sigma_j\sigma_i & \text{se } |i-j| > 1 \\ \sigma_i\sigma_j\sigma_i = \sigma_j\sigma_i\sigma_j & \text{se } |i-j| = 1 \end{array} \right\rangle$$

In [GGM10a, Proposizione 9] viene mostrato che per $\delta_n = \sigma_{n-1} \dots \sigma_1 \in B_n$ vale $|SC(\delta_n)| = 2^{n-2} - 2$.

n	4	5	6	7	8	9	10	11	12
$\lvert SC(\delta_n)\rvert$	2	6	14	30	62	126	254	510	1022

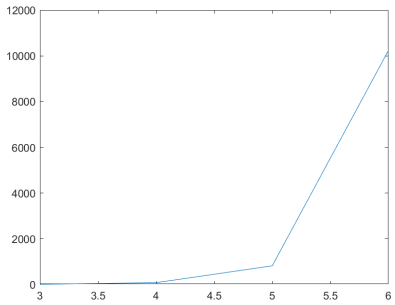
Ne deduciamo che la complessità dell'algoritmo proposto non può essere polinomiale. Infatti la soluzione del problema può richiedere il calcolo di tutto l'insieme degli sliding circuits, che come illustra questo esempio può essere esponenziale in n.



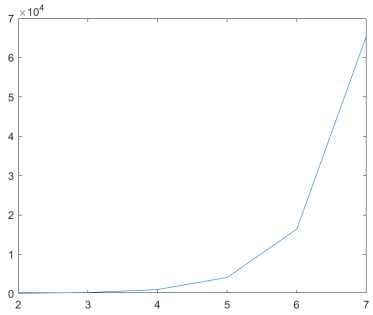
Un esempio più problematico viene mostrato in [GM11, Proposizione 5.5]. Per $k \geq 1$ e $n \geq 3$ consideriamo la treccia $\beta_{k,n} \in B_{n+2}$ data da

$$\beta_{k,n} = (\sigma_1 \dots \sigma_{n-1})^{nk+1} (\sigma_{n+1})^{4k+1}$$

k	1	1	1	1
n	3	4	5	6
$\lvert SC(\beta_{k,n})\rvert$	12	84	820	10232



k	2	3	4	5	6	7
n	3	3	3	3	3	3
$\lvert SC(\beta_{k,n})\rvert$	60	252	1020	4092	16380	65532



Infatti si ha che la lunghezza canonica di $\beta_{k,n}$ è $\ell(\beta_{k,n}) = 4k + 1$ e che l'insieme degli sliding circuits di $\beta_{k,n}$ ha cardinalità esponenziale in $\ell(\beta_{k,n})$ e n. Più precisamente vale la stima $|SC(\beta_{k,n})| \geq 2^{n-2}(n-1)^{2k-1}$. Riportiamo due tabelle con la cardinalità di $|SC(\beta_{k,n})|$ per alcuni valori di k, n con il grafico corrispondente per visualizzarne meglio la crescita esponenziale (non sono stati fatti ulteriori test a causa del tempo di calcolo che hanno richiesto questi, diverse ore per alcuni elementi)

Diamo adesso un'approssimazione sul valore medio della dimensione dell'insieme degli sliding circuits su un campione casuale di trecce. Sono state presi in esame insiemi di 1000 elementi di B_n per $n = 3, 4, 5, 6$ generati casualmente come prodotto di $k = 10, 50, 100$ elementi tra i generatori di B_n e i loro inversi (la scelta di un campione così piccolo è stata nuovamente motivata dal tempo necessario per eseguire tutti i calcoli). L'ordine di moltiplicazione dei generatori e dei loro inversi è stato scelto casualmente con distribuzione uniforme.

k	10	10	10	10
n	3	4	5	6
$\lvert SC(\beta_{k,n})\rvert$	5.7160	8.5760	12.8860	25.0220

k	50	50	50	50
n	3	4	5	6
$\lvert SC(\beta_{k,n})\rvert$	25.9390	26.2580	23.8480	23.6160

k	100	100	100	100
n	3	4	5	6
$\lvert SC(\beta_{k,n})\rvert$	50.4270	50.4820	44.5280	39.6460

Anche se il campione preso in considerazione è abbastanza piccolo, e quindi la media non precisa, ricaviamo alcune informazioni relative al comportamento di SC. In particolare vediamo come la dimensione dell'insieme SC sia generalmente contenuta rispetto a n. Nonostante la teoria assicuri l'esistenza di casi patologici questi sembrano non essere troppo frequenti all'interno del gruppo di trecce, rendendo l'algoritmo utilizzabile nella pratica.

Riferimenti bibliografici

[EM94] Elsayed A. Elrifai and Hugh R. Morton, *Algorithms for positive braids*, The Quarterly Journal of Mathematics **45** (1994), no. 4, 479–497.

[Gar69] Frank A. Garside, *The braid group and other groups*, The Quarterly Journal of Mathematics **20** (1969), no. 1, 235–254.

[GGM10a] Volker Gebhardt and Juan González-Meneses, *The cyclic sliding operation in Garside groups*, Mathematische Zeitschrift **265** (2010), 85–114.

[GGM10b] ———, *Solving the conjugacy problem in Garside groups by cyclic sliding*, Journal of Symbolic Computation **45** (2010), no. 6, 629–656.

[GM11] Juan González-Meneses, *On reduction curves and garside properties of braids*, Topology of Algebraic Varieties and Singularities: Conference in Honor of Anatoly Libgober's 60th Birthday, June 22-26, 2009, Jaca, Huesca, Spain, vol. 538, American Mathematical Soc., 2011, p. 225.