ENLACEMENTS ET TRESSES FERMEES
TRAITEMENT ALGORITHMIQUE

Alain JACQUEMARD
Laboratoire de Topologie UA 755
BP 138
21004 Dijon CEDEX

Introduction.

Depuis fort longtemps, on cherche une solution algorithmique au problème de la classification des enlacements. Une méthode d'approche consiste à tenter de classifier les mots de tresse, qui apparaissent naturellement lorsqu'on effectue certaines transformations géométriques sur les enlacements. Pour classifier ces mots de tresse il faut en particulier être capable de résoudre le problème de la conjugaison dans les groupes de tresses à nombre de brins fixé. F.Garside, puis J.S.Birman donnèrent des algorithmes pour traiter ce problème, en utilisant une notion de forme normale de mot de tresse, mais il ne s'intéressèrent pas au problème du calcul effectif pour des nombres de brins supérieurs à trois, le calcul manuel devenant impossible. Sur machine aussi, ces algorithmes ne sont pas assez efficaces, et c'est pourquoi on montre ici qu'on peut en fait calculer les formes normales de Garside avec une complexité cubique en la longueur du mot de tresse. Une conséquence pratique de ceci est de pouvoir maintenant classifier effectivement les classes de conjugaison des tresses à nombre de brins allant jusqu'à cinq, en donnant à chaque fois un représentant normal unique pour la classe.

I Enlacements et tresses fermées.

On appelle *enlacement* (linéaire par morceaux) de la sphère $S_3 = R^3 \cup \{\infty\}$ toute union de polygones disjoints plongés dans S_3.

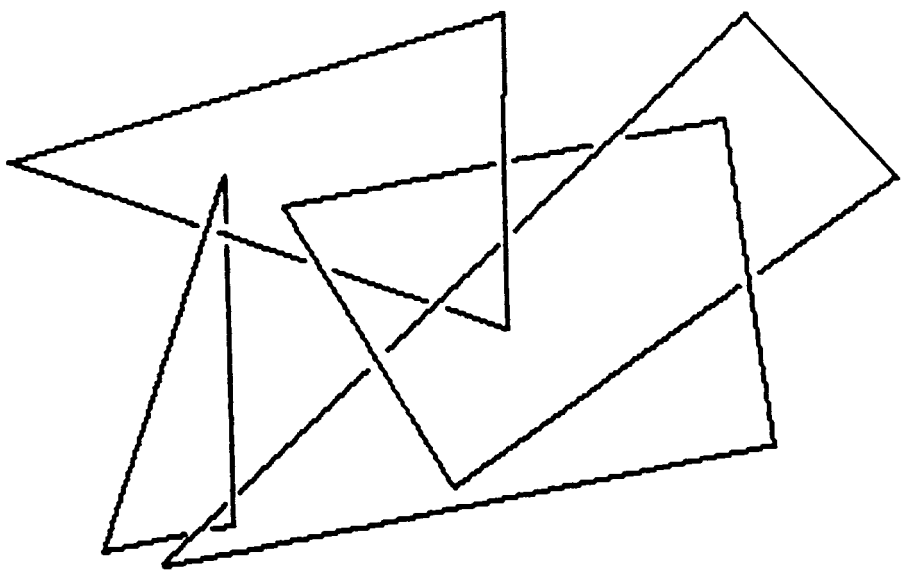

Figure 1

On dit que deux enlacements L_1 et L_2 sont *isotopes* s'il existe un homéomorphisme h de S_3 tel que $h(L_1) = L_2$. On connaît de nombreux invariants d'isotopie (cf [Ro],[BZ]) concernant les enlacements, soit numériques (nombre de composantes connexes, nombre d'enlacement...), soit algébriques (polynômes d'Alexander, Jones...), qui peuvent permettre de commencer une classification, mais qui sont cependant insuffisants pour caractériser une classe d'isotopie donnée.

C'est pourquoi on préfère considérer les enlacements sous forme géométrique "normale", dite en *tresse fermée*, et surtout l'expression algébrique qui lui est attachée, le *mot de tresse*, auquel on ne peut associer qu'une seule classe d'isotopie. Nous verrons dans les paragraphes suivants les difficultés que pose cette méthode de classification.

Pour l'instant définissons de manière précise en quoi consiste cette forme géométrique "normale", ainsi que la notion de groupe des tresses. Précisément, étant donnés un axe

orienté ℓ, et un plan P orienté orthogonal à ℓ passant par une origine O commune à ℓ et P, considérons la projection $\tilde{L}$ de l'enlacement L sur P, que nous appellerons représentation plane de L. Nous supposerons de plus que L est en position générique par rapport à P, c'est à dire qu'aucun segment de L n'est colinéaire à ℓ. Ainsi aucun segment de L ne se trouve représenté par un point sur P, et il correspond à tout segment de la représentation plane un segment et un seul de l'enlacement L. Cette hypothèse n'est pas restrictive car un théorème de Markov (cf [Bi] p 39) assure qu'on peut toujours isotoper un enlacement en position non générique en un qui le soit.

Orientons maintenant la représentation $\tilde{L}$. Certains segments de $\tilde{L}$ peuvent alors être orientés positivement eu égard à l'orientation de P, d'autres négativement. Dans le cas où tous les segments de la représentation plane $\tilde{L}$ sont orientés positivement, on dira que relativement à l'axe ℓ et aux orientations, L se présente sous forme de *tresse fermée* ou sous forme géométrique normale.

La tresse fermée ainsi obtenue est telle que toute demi-droite de P passant par O rencontre la représentation plane $\tilde{L}$ en un nombre constant de points (en comptant la multiplicité d'intersection). Ce nombre est le nombre de brins de la tresse. Le nombre minimal de brins nécéssaire pour representer en tresse fermée un type isotopique est nommé l'index. On ne sait pas calculer, ni caractériser cet index. Cependant V.F.R. Jones a donné une inégalité entre l'index et la valeur de son polynôme qui se révèle extrêmement fine (cf [Jo]).

Les tresses à n brins forment en fait un groupe noté $B(n)$ engendré par $n-1$ générateurs $\sigma_1 \ldots \sigma_{n-1}$, muni des relations:

$$rc_{i,j} \qquad \sigma_i\sigma_j = \sigma_j\sigma_i \qquad si \quad |i-j|>1$$

$$rg_i \qquad \sigma_i\sigma_{i+1}\sigma_i = \sigma_{i+1}\sigma_i\sigma_{i+1} \qquad si \quad i<n-1$$

Pour avoir une idée précise de la loi de groupe, il faut savoir qu'on peut définir $B(n)$ (entre autres...) comme le groupe fondamental $\pi_1((E^n - \mathcal{D}_n)/S(n))$ où:
E désigne le plan euclidien, $S(n)$ le groupe symétrique,
$\mathcal{D}_n = \{(x_1, x_2, \ldots, x_n)/ \quad \exists\,(i,j) \quad , i \neq j, \quad tq \quad x_i = x_j\}$.
Consulter la figure suivante pour voir un représentant du ième générateur.

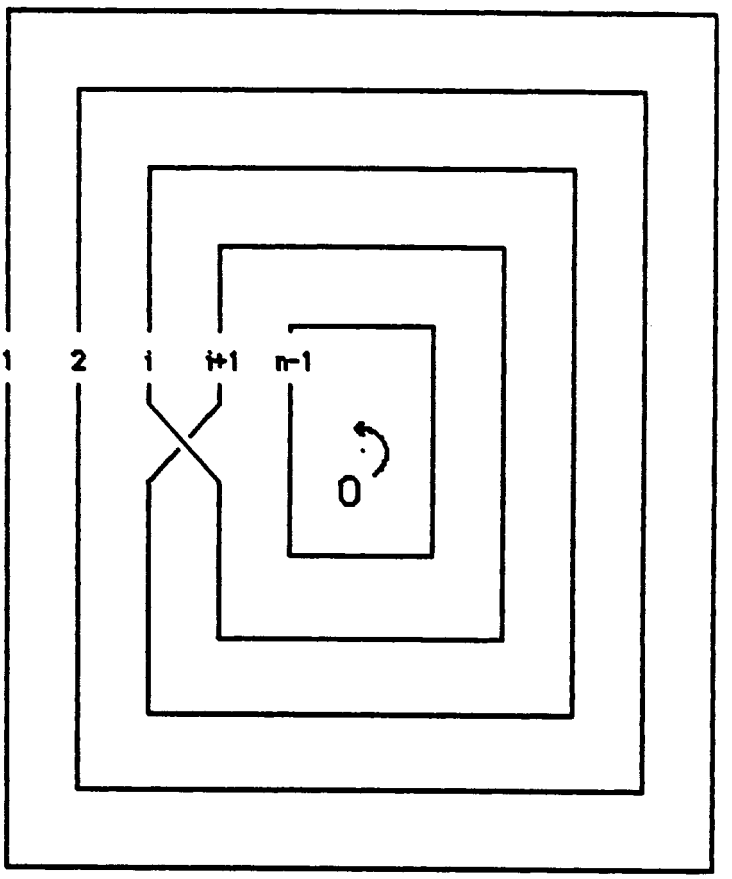

Figure 2

Pour lire le mot de tresse correspondant à une tresse fermée donnée sous forme de représentation plane, il suffit de regarder en tournant dans le sens direct chaque intersection et de faire apparaître le générateur σ_i si le $i$ème brin passe au dessus du $i+1$ème et σ_i^{-1} dans le cas contraire (voir par exemple la Figure 4) .

On dispose de plusieurs méthodes pour déformer algorithmiquement L de façon à le mettre sous forme de tresse fermée. La plus ancienne et la plus simple à concevoir est celle d' Alexander, énoncée en 1923, en tant que preuve du

THÉORÈME. *Etant donné un enlacement L en position générique par rapport à un axe ℓ, les orientations étant précisées, on peut l'isotoper par une suite finie de transformations élémentaires à une tresse fermée.*

Les transformations élémentaires en question sont des dents de scie, qui consistent à remplacer un segment dont la projection est mal orientée sur la représentation plane par deux segments qui par exemple passent "au-dessus" de tout le reste de l'enlacement, ainsi qu'on peut le constater sur la figure suivante, où le segment mal orienté [A,B] est remplacé par les deux segments [A,C] et [C,B].

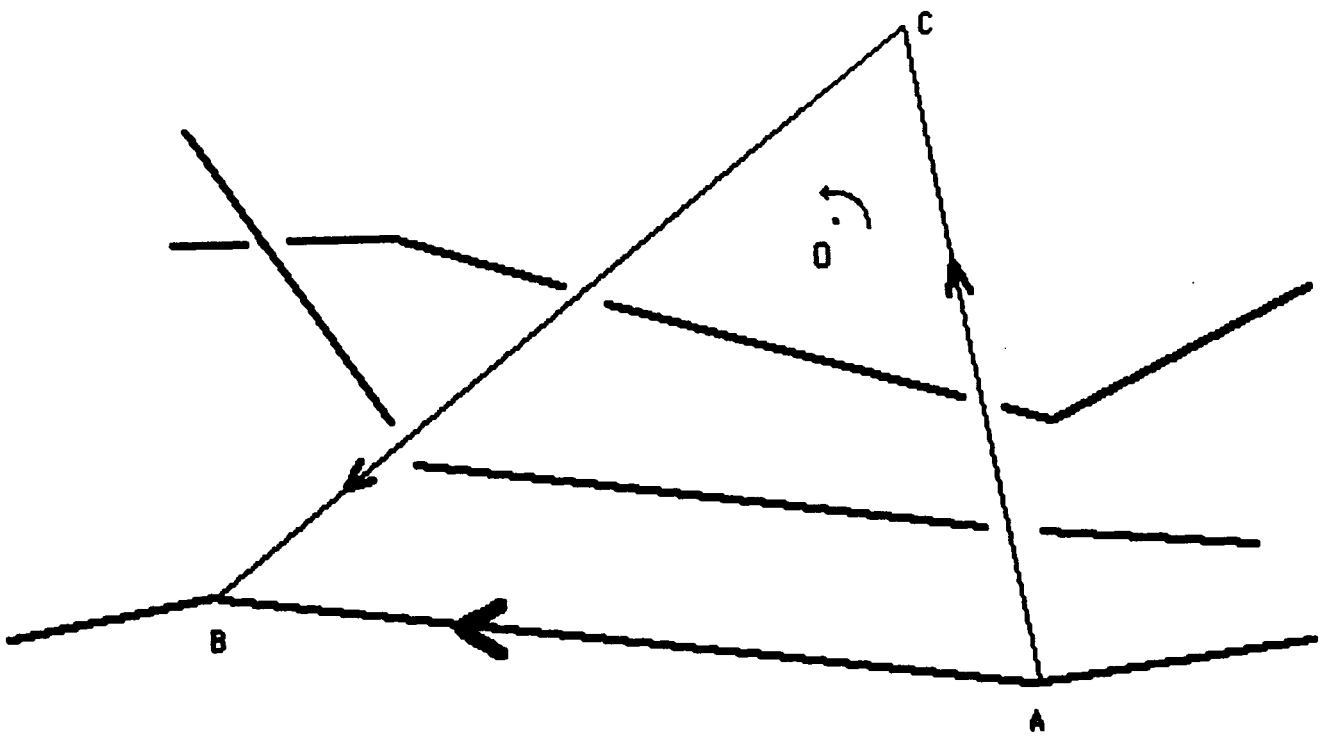

Figure 3

On a implémenté cette méthode sur micro-ordinateur, mais du fait du nombre important de paramètres à gérer pour le choix des dents de scie, ce programme est d'un usage pratique parfois délicat.

Mais une bien meilleure méthode consiste à utiliser un autre algorithme donné en 1987 par S.Yamada, et qui est basé sur la notion de cercle de Seifert. Sa formulation mathématique est plus élaborée que celle de Markov, mais le programme qui en résulte est très efficace. Sur ce sujet, nous renvoyons le lecteur intéressé à [Ya] pour la théorie et à [Me] pour la partie programmation.

II Type isotopique et tresses fermées.

Maintenant que nous savons associer à tout enlacement un mot de tresse, il nous faut répondre à deux questions essentielles :

comment varie ce mot si on change l'enlacement en un enlacement isotope ?

comment reconnaître à partir de deux mots de tresse s'ils représentent la même classe d'isotopie d'enlacements ?

Les réponses sont contenues dans le théorème suivant, dû à Markov:

THÉORÈME. *Deux mots de tresse ω_1 et ω_k représentent des enlacements isotopes s'il existe une chaîne $\omega_1 \rightarrow \omega_2 \ldots \rightarrow \omega_k$ de tresses fermées , deux tresses fermées consécutives dans la chaîne ω_i et ω_j étant telles que :*

ou bien elles ont même nombre de brins n et sont conjuguées[1] *dans le groupe $B(n)$;*

ou bien $\omega_j = \omega_i \sigma_{n-1}^{\epsilon}$ avec ω_i ne contenant que des générateurs d'indice inférieurs à $n-2$, ϵ valant 1 ou -1 ;

Nous donnerons ici seulement la réponse algorithmique au problème de la conjugaison à nombre de brins constant, le problème complet avec augmentations et/ou diminutions du nombre de brins n'étant jusqu'ici pas encore résolu théoriquement. On sait cependant qu'on ne peut s'affranchir de telles transformations puisqu'il existe des tresses de $B(4)$ dont aucun conjugué dans $B(4)$ n'est réductible (c'est à dire où le générateur σ_3 n'apparaît qu'une seule fois, à une puissance positive ou négative) et qui pourtant représentent le nœud trivial. H.R.Morton donne en particulier l'exemple de la tresse $\omega_1 = \sigma_3^{-2}\sigma_2\sigma_3^{-1}\sigma_2\sigma_1^3\sigma_2^{-1}\sigma_1\sigma_2^{-1}$ qui est irréductible. Mais il lui ajoute σ_4, obtenant ainsi un représentant ω_1^* dans $B(5)$ de la même classe d'isotopie. Ensuite, il prend un autre élément ω_2^* de la classe de conjugaison de ω_1^* dans $B(5)$, qui se réduit alors dans $B(4)$ en la tresse $\omega_2 = \sigma_2^{-1}\sigma_3^{-1}\sigma_2\sigma_3^{-2}\sigma_2\sigma_1^3\sigma_2^{-1}\sigma_1$. Or celle-ci se réduit en une tresse de $B(3)$, puis en une tresse de $B(2)$, rprésentant le mot trivial. On se reportera à [Mo] pour les détails de la démonstration. Pour en finir avec cet exemple, on verra plus tard qu'on peut prouver par le calcul sur ordinateur que les tresses de ω_1 et ω_2, qui sont dans la même classe d'isotopie, ne sont pas conjuguées dans $B(4)$.

[1] *ie* il existe $\alpha \in B(n)$ telle que $\omega_i = \alpha^{-1}\omega_j\alpha$

Il semble clair que la détermination par un algorithme d'un mot de tresse canonique pour chaque type isotopique est lié à l'existence d'un théorème garantissant que le nombre de ces transformations du deuxième type est fini.

L'étape primordiale dans la solution du problème de la conjugaison dans $B(n)$ consiste d'abord à savoir trouver pour tout mot une écriture équivalente en tant qu'élément du groupe $B(n)$, écriture certes insuffisante pour décrire exactement la classe d'isotopie, mais qui de par sa forme nous donne des premiers renseignements. C'est ce rôle que va jouer la forme normale de Garside, ainsi que nous allons le voir dans le paragraphe suivant.

III La solution de Garside au problème du mot dans le groupe $B(n)$.

Dans toute la suite, on fera la distinction entre l'écriture d'un mot de tresse, c'est à dire l'élément du groupe libre engendré par $\sigma_1, \sigma_2, \ldots, \sigma_{n-1}$ et la tresse fermée qu'il représente dans $B(n)$. On notera par le signe $=$ l'égalité dans le groupe libre (les écritures), et par $\equiv$ l'égalité dans $B(n)$. Nous allons commencer par définir la forme normale d'un mot en décrivant simultanément son algorithme original d'obtention dû à Garside. Avant celà il nous faut définir la notion de *diagramme* d'un mot de tresse ω *positif*, c'est à dire ne comportant que des puissances positives des générateurs : posons $D_0(\omega) = \{\omega\}$ et soit $D_{i+1}(\omega)$ l'ensemble constitué des mots de $D_i(\omega)$ ainsi que ceux obtenus en appliquant l'une quelconque des relations $rc_{i,j}$ ou rg_i aux éléments de $D_i(\omega)$. Puisque par application des relations on ne modifie pas la longueur des mots, il existe un indice m pour lequel $D_m(\omega)$ est maximal, et $D_m(\omega)$ est par définition le diagramme $D(\omega)$.

D'autre part, un mot de tresse, le mot $\Delta_n = \sigma_1\sigma_2\ldots\sigma_{n-1}\sigma_1\sigma_2\ldots\sigma_{n-2}\ldots\sigma_1\sigma_2\sigma_1$ possède deux propriétés particulièrement intéressantes:

1) pour chaque générateur σ_i il existe un mot positif que nous noterons τ_i tel que $\sigma_i^{-1} = \Delta_n^{-1}\tau_i$,

2) pour tout générateur σ_i, $\Delta_n^{-1}\sigma_i = \sigma_{n-i}\Delta_n^{-1}$

Ces deux propriétés vont nous permettre d'éliminer les puissances négatives isolées dans ω et de travailler ensuite avec seulement des mots positifs.

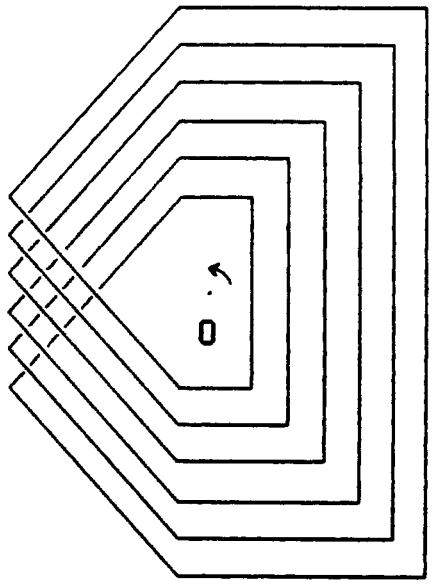

Figure 4 : le mot Δ_5

Ceci posé, l'algorithme classique d'obtention de la forme normale d'un mot de tresse ω est le suivant :

phase 1 : dans ω on remplace chaque σ_i^{-1} par $\Delta_n^{-1}\tau_i$.

phase 2 : on fait circuler les Δ_n^{-1} à gauche en utilisant le fait que

$$\Delta_n^{-1}\sigma_i = \sigma_{n-i}\Delta_n^{-1} \ ;$$

on obtient ainsi $\omega \equiv \Delta_n^k \omega^+$ avec ω^+ mot positif.

phase 3 : on calcule $D(\omega^+)$ et on y choisit un élément de la forme $\Delta_n^m \bar{\omega}$ avec m maximal.

phase 4 : on calcule enfin $D(\bar{\omega})$ et on y choisit l'écriture minimale ω^* pour l'ordre lexicographique.

Ainsi on obtient finalement $\omega \equiv \Delta_n^{k+m}\omega^*$ et cette forme d'écriture unique est appelée forme normale de Garside du mot ω. L'entier $k+m$ est appelé puissance de Garside de ω et ω^* est appelé reste de Garside.

Une implémentation de cet algorithme a été bien sûr réalisée, mais le temps de calcul du diagramme $D(\omega)$ est très grand : il suffit de remarquer que le diagramme du mot Δ_4 possède déjà 768 éléments ... Ainsi, le programme ne permet pas d'aller beaucoup plus loin que le calcul manuel, et surtout la complexité croit de manière énorme aussi bien avec la longueur du mot à traiter, qu'avec le nombre de brins.

C'est pourquoi nous décrivons dans le paragraphe suivant un nouvel algorithme, qui est de complexité acceptable.

IV Un algorithme d'extraction.

Nous allons maintenant montrer comment on peut améliorer les phases 3 et 4 de l'algorithme de Garside en évitant de calculer tout le diagramme. Pour celà on va introduire un nouvel algorithme qui est capable de déterminer si un générateur donné est extractible à gauche d'un mot de tresse *positif*, et si oui, de donner la chaîne de réécritures conduisant à l'extraction.

Donnons d'abord les définitions suivantes, valables pour des mots positifs:

DÉFINITION 1. *on dit qu'un générateur σ_i est extractible à gauche de ω si ω peut s'écrire $\sigma_i\tilde{\omega}$.*

DÉFINITION 2. *on appelle indice de blocage à gauche $I^g_{\sigma_i}(\omega)$ du générateur σ_i dans l'écriture de ω le nombre de générateurs σ_{i+1} et σ_{i-1} situés entre le premier générateur de ω et le premier σ_i apparaissant à gauche dans l'écriture de ω.*

Bien noter que cette définition dépend de l'écriture du mot et non pas de l'élément de $B(n)$ qu'elle représente.

Pour trouver la puissance maximale de Δ_n présente dans les écritures de ω nous allons extraire successivement chaque Δ_n, et pour ce faire nous allons extraire successivement chaque générateur présent dans Δ_n dans l'ordre où il apparaît.

La procédure d'extraction à gauche d'un générateur σ_i consiste à tenter d'abord d'extraire directement σ_i, ou sinon d'effectuer un *glissement à gauche* c'est à dire une transformation du type $\alpha\sigma_j\sigma_i\sigma_j\beta \rightarrow \alpha\sigma_i\sigma_j\sigma_i\beta$, avec $j = i+1$ ou $j = i-1$, et de réappliquer de nouveau la procédure d'extraction.

Explicitons cet algorithme:

phase $\mathcal{M}_g$: faisons migrer à gauche le premier σ_i présent à gauche de l'écriture de ω en utilisant récursivement les règles de réécriture issues des relations $rc_{k,i}$: $\sigma_k\sigma_i \rightarrow \sigma_i\sigma_k$ si $|k-i|>1$. Notons ω^* l'écriture obtenue à l'aide de ces transformations. Si σ_i apparaît en tête de ω^*, c'est terminé.
Sinon on passe à la phase suivante.

phase $\mathcal{G}_g$: Si $I^g_{\sigma_i}(\omega^*) > 0$. Dans ce cas le premier σ_i est bloqué à gauche par un σ_j avec $j=i+1$ ou $j=i-1$, et ω^* s'écrit donc :

$\omega^* = \alpha_1\sigma_j\sigma_i\alpha_2$ avec α_1 ne contenant pas σ_i.
On appelle alors la procédure d'extraction du générateur σ_j sur le mot α_2.

- Si l'extraction de σ_j du mot α_2 est impossible alors l'extraction de σ_i du mot ω se termine par une impossibilité.

- Dans le cas contraire, on écrit ω^* sous la forme $\alpha_1\sigma_j\sigma_i\sigma_j\bar{\alpha}_2$ et par application de la relation rg_i ou rg_j on obtient l'écriture $\omega^{**} = \alpha_1\sigma_i\sigma_j\sigma_i\bar{\alpha}_2$ et ainsi $I^g_{\sigma_i}(\omega^{**}) < I^g_{\sigma_i}(\omega^*)$. On effectue alors la procédure d'extraction à gauche de σ_i sur ω^{**}.

D'après la présentation ci-dessus, l'algorithme converge puisqu'à chaque appel de procédure d'extraction, soit on diminue la longueur de l'écriture, soit on reste avec une écriture de même longueur mais alors on diminue l'indice de blocage. A chaque étape $\mathcal{G}_g$ l'indice de blocage décroit strictement, donc on parvient soit à l'extraction du générateur soit à la preuve qu'il n'est pas extractible.

Le seul point qui reste à vérifier est l'affirmation de la phase $\mathcal{G}_g$ selon laquelle l'impossibilité de l'extraction de σ_j du mot α_2 entraine l'impossibilité de l'extraction de σ_i du mot ω. En effet, dans ce cas il peut se faire que l'indice de blocage par rapport à σ_i ne soit pas minimal. C'est le cas si ω peut s'écrire $\omega = \bar{\alpha}_1\sigma_k\sigma_j\sigma_i\alpha_2$ avec $|k-i|=1$, $k\neq j$, et σ_k extractible de α_2: $\alpha_2 = \sigma_k\bar{\alpha}_2$. Nous allons montrer qu'alors σ_j constitue un obstacle infranchissable pour le σ_i. Diminuons donc l'indice de blocage par rapport à σ_i en réécrivant ω sous la forme $\tilde{\omega}^* = \alpha_1\sigma_j\sigma_i\sigma_k\sigma_i\bar{\alpha}_2$. Mais σ_j n'est pas extractible de $\sigma_k\sigma_i\bar{\alpha}_2$ car il faudrait pour celà que σ_j soit extractible de $\sigma_i\bar{\alpha}_2$ et ceci ne pourrait avoir lieu que si déjà σ_j était extractible de $\bar{\alpha}_2$. Mais alors, puisque $|k-j|=2$, σ_j serait extractible de $\sigma_k\bar{\alpha}_2 = \alpha_2$, ce qui n'est pas.

Ainsi, si on peut diminuer l'indice de blocage $I^g_{\sigma_i}$ par une réécriture faisant intervenir σ_k le nouveau mot $\tilde{\omega}^* = \alpha_1\sigma_j\sigma_i\tilde{\alpha}_2$ obtenu est tel que σ_j n'est pas extractible de $\tilde{\alpha}_2$. Par récurrence, l'indice de blocage $I^g_{\sigma_i}$ ne peut jamais être nul au cours des diverses réécritures obtenues à partir de ω. L'algorithme énoncé répond donc au problème posé.

Evaluons la complexité de cet algorithme d'extraction. Nous définissons comme opération de complexité élémentaire toute comparaison entre deux générateurs ou tout échange de générateurs dans une écriture. Notons alors $c(l)$ le nombre de telles opérations élémentaires nécéssaires à l'extraction d'un générateur d'un mot de longueur l. Supposons qu'à l'étape M_g , pour faire migrer σ_i on doive faire k comparaisons et donc au plus $k-1$ échanges. Si la migration conduit à un blocage on doit alors appeler la procédure d'extraction avec un mot de longueur $l-k-1$ d'où une complexité $c(l-k-1)$. On en déduit que $c(l) \leq 2k-1+c(l-k-2)$. Ce qui conduit à $c(l) = O(l^2)$. Il en est de même pour l'algorithme d'extraction complète de Δ_n puisque la longueur de ce mot est fixe.

Regardons sur un exemple explicite comment fonctionne l'extraction: soit par exemple à extraire à gauche le générateur σ_2 du mot $\omega = \sigma_3\sigma_3\sigma_1\sigma_4\sigma_2\sigma_1\sigma_3\sigma_4\sigma_2\sigma_2\sigma_3\sigma_1$ La phase de glissement donne $\omega' = \sigma_3\sigma_3\sigma_1\sigma_2\sigma_4\sigma_1\sigma_3\sigma_4\sigma_2\sigma_2\sigma_3\sigma_1$. L'indice de blocage $I^g_{\sigma_2}(\omega')$ est 3. Comme σ_2 n'est pas en tête, on cherche à extraire σ_1 de $\alpha_2 = \sigma_4\sigma_1\sigma_3\sigma_4\sigma_2\sigma_2\sigma_3\sigma_1$. Après glissement on obtient $\alpha'_2 = \sigma_1\sigma_4\sigma_3\sigma_4\sigma_2\sigma_2\sigma_3\sigma_1$. On remonte alors à l'extraction de σ_2 du mot ω' qui s'écrit maintenant, après glissement, $\omega'' = \sigma_3\sigma_3\sigma_2\sigma_1\sigma_2\sigma_4\sigma_3\sigma_4\sigma_2\sigma_2\sigma_3\sigma_1$.
L'indice de blocage $I^g_{\sigma_2}(\omega'')$ est 2.
Il faut maintenant extraire σ_3 de $\sigma_1\sigma_2\sigma_4\sigma_3\sigma_4\sigma_2\sigma_2\sigma_3\sigma_1$.
Donc extraire σ_3 de $\sigma_1\sigma_2\sigma_3\sigma_4\sigma_3\sigma_2\sigma_2\sigma_3\sigma_1$, extraire σ_2 de $\sigma_4\sigma_3\sigma_2\sigma_2\sigma_3\sigma_1$, extraire σ_3 de $\sigma_2\sigma_3\sigma_1$, et enfin σ_2 de σ_1 ce qui est impossible.
L'extraction de σ_2 de $\sigma_3\sigma_3\sigma_1\sigma_4\sigma_2\sigma_1\sigma_3\sigma_4\sigma_2\sigma_2\sigma_3\sigma_1$ est donc impossible.

Décrivons maintenant comment on peut améliorer la phase 4.
Pour construire l'écriture minimale ω^* de ω suivant l'ordre lexicographique on va utiliser aussi l'algorithme d'extraction. En effet, on cherche par essais successifs sur σ_1,σ_2,..., le plus petit générateur extractible à gauche σ_k. Ainsi ω s'écrit $\sigma_k\bar{\omega}$, et alors ω^* s'écrit $\sigma_k\bar{\omega}^*$.
Mais pour calculer $\bar{\omega}^*$ on peut se contenter de chercher à extraire des générateurs d'indice

au moins $k-1$, puisque sinon on pourrait par application d'une règle $rc_{i,j}$ faire parvenir ce générateur en tête de ω et σ_k ne serait pas minimal.

La complexité est ici plus élevée que dans la phase précédente puisqu'on doit au pire essayer les $n-1$ générateurs pour chaque position de l'écriture. La complexité est donc en $O(l^3)$ pour cette phase 4.

La complexité totale de cet algorithme de mise sous forme de Garside est donc elle aussi cubique en la longueur du mot, compte tenu du fait qu'à la suite de la phase 1 la longueur du mot à traiter est au plus de $l \times longueur(\Delta_n)$.

V Le problème de la conjugaison dans le groupe $B(n)$.

Le problème de la conjugaison dans $B(n)$ se résoud lui aussi de manière algorithmique, mais la complexité reste exponentielle par rapport à la complexité du calcul de la forme de Garside.

Rappelons que deux tresses ω_1 et ω_2 sont conjuguées dans $B(n)$ s'il existe une tresse α dans $B(n)$ telle que $\omega_1 = \alpha^{-1}\omega_2\alpha$, et définissons l'ensemble $I(n)$ des formes initiales de Δ_n:

$I(n) = \{\alpha^* \quad / \quad \alpha \in B(n) \quad / \quad \exists\beta \in B(n) \quad / \quad \exists\delta \in D(\Delta_n) \quad / \quad \alpha\beta = \delta\}$

où α^* désigne la forme minimale de α sous l'ordre lexicographique.

L'interêt de $I(n)$ est fondamental car il suffit de regarder les conjugaisons par ses éléments pour connaître tous les conjugués d'un mot (cf [Bi]).

Définissons $S_0(\omega) = \{\omega\}$, et d'une manière générale soit $S_{i+1}(\omega)$ l'ensemble constitué des éléments de $S_i(\omega)$ et des conjugués des éléments de $S_i(\omega)$ par les éléments de $I(n)$, de puissance de Garside supérieure où égale à celle de ω. Puisque l'ensemble des conjugués de ω de puissance de Garside supérieure à un entier fixé est fini, il existe un entier m pour lequel $S_m(\omega)$ est maximal. Notons enfin $S(\omega)$ l'ensemble des éléments de puissance de Garside maximale de $S_m(\omega)$. Parmi les éléments de $S(\omega)$ considérons celui qui s'écrit $\Delta^s\omega^{**}$ avec ω^{**} minimal pour l'ordre lexicographique. L'entier s est la puissance sommitale et ω^{**} est le reste sommital de ω. On a le:

THÉORÈME. *: s et ω^{**} sont caractéristiques de la classe de conjugaison de ω dans $B(n)$.*

Les démonstrations sont dans [Bi] et dans [Ga] .

Il en résulte un premier algorithme de construction de la forme sommitale de ω_0 :
1) on calcule la forme de Garside $\Delta_n^g \omega_0^*$ de ω_0, on range cette forme en mémoire.
2) pour toute forme $\Delta_n^g \omega^*$ présente en mémoire,
pour toute forme initiale α on calcule

- si g est pair la forme de Garside de $\alpha^{-1}\omega^*\alpha$

- si g est impair la forme de Garside de $\bar{\alpha}^{-1}\omega^*\alpha$ où $\bar{\alpha}^{-1}$ est l'image de α^{-1} par le morphisme de groupe qui envoie σ_i sur σ_{n-i} pour tout $i \in [1, n-1]$

2.1) si cette puissance de Garside est identique à celle de ω_0, on range cette nouvelle forme en mémoire.

2.2) si cette puissance de Garside est strictement supérieure, on efface la mémoire et on recommence tout en 1) avec ce nouveau mot.
3) quand toutes les formes présentes en mémoire ont été utilisées, on prend celle qui présente un reste minimal pour l'ordre lexicographique.

La forte complexité de cette méthode est dûe au nombre de formes initiales de Δ_n, qui croit très vite avec n. Par exemple, pour $n = 4$, on compte 6 formes initiales, mais pour $n = 5$, il y en a déjà 119.

On terminera le paragraphe en indiquant qu'on peut mettre en œuvre un algorithme légèrement plus rapide (on a noté un gain d'environ un cinquième du temps de calcul). Cet algorithme repose sur l'observation suivante:

LEMME. *si ω s'écrit sous la forme $\omega = \gamma^-\gamma^+$ avec γ^+ positif et γ^- négatif, sa puissance de Garside est positive si et seulement si $(\gamma^-)^{-1}$ est extractible à gauche de γ^+.*

preuve: seule la condition nécessaire présente de l'interêt. Supposons que $(\gamma^-)^{-1}$ ne soit pas extractible de γ^+. Ceci signifie que l'on peut écrire après des premieres simplifications éventuelles $\omega = \alpha^-\sigma_k^{-1}\beta^+$ avec σ_k non extractible à gauche de β^+ et α^- négatif. Supposons maintenant que ω s'écrive $\Delta_n^g\omega^*$ avec $g > 0$. Mais alors $\sigma_k^{-1}\beta^+$ s'écrit $(\alpha^-)^{-1}\Delta_n^g\omega^*$ qui est positif, et ainsi σ_k serait extractible de β^+ à gauche.

Finalement, l'algorithme que l'on a retenu pour l'implémentation s'énonce ainsi:

1) on calcule la forme de Garside $\Delta_n^g \omega_0^*$ de ω_0, on range cette forme en mémoire.

2) pour toute forme $\Delta_n^g \omega^*$ présente en mémoire,

pour toute forme initiale α,

2.1 si g est pair on écrit $\alpha^{-1}\omega^*\alpha = \gamma^-\gamma^+$ avec γ^- négatif et γ^+ positif.

2.1.1 si $(\gamma^-)^{-1}$ n'est pas extractible à gauche de γ^+, alors on recommence immédiatement en 2) en prenant une nouvelle forme initiale.

2.1.2 si $(\gamma^-)^{-1}$ est extractible à gauche de γ^+, on effectue cette extraction, on calcule la forme de Garside $\Delta_n^h r^*$ du résultat obtenu. Si h est nul, on range la forme $\Delta_n^g r^*$ en mémoire; si $h > 0$, on efface la mémoire et on recommence tout en 1) avec le mot $\Delta_n^{g+h} r^*$.

2.2 si g est impair on écrit $\bar{\alpha}^{-1}\omega^*\alpha = \gamma^-\gamma^+$ avec γ^- négatif et γ^+ positif, $\bar{\alpha}^{-1}$ désignant l'image de α^{-1} par le morphisme de groupe qui envoie σ_i sur σ_{n-i} pour tout $i \in [1, n-1]$. On traite maintenant ce cas de manière similaire à 2.1 .

3) quand toutes les formes présentes en mémoire ont été utilisées, on prend celle qui présente un reste minimal pour l'ordre lexicographique.

A partir de $n = 6$, tout calcul devient impossible: le mot Δ_6 est de longueur 15, et le nombre de formes initiales est déjà si important que leur calcul se révèle impossible. A titre d'illustration de l'utilisation de ce logiciel, on peut confirmer que les deux tresses de l'exemple de Morton évoqué précédemment ne sont pas conjugués dans $B(4)$: on a calculé en effet que la forme sommitale de $\omega_1 = \sigma_3^{-2}\sigma_2\sigma_3^{-1}\sigma_2\sigma_1^3\sigma_2^{-1}\sigma_1\sigma_2^{-1}$ est
$\Delta_4^{-3}\sigma_1^4\sigma_2\sigma_1^4\sigma_2\sigma_3\sigma_2^2\sigma_1^2\sigma_3\sigma_2^2\sigma_1$ et que celle de $\omega_2 = \sigma_2^{-1}\sigma_3^{-1}\sigma_2\sigma_3^{-2}\sigma_2\sigma_1^3\sigma_2^{-1}\sigma_1$ est
$\Delta_4^{-3}\sigma_1^4\sigma_2\sigma_1^4\sigma_2\sigma_3\sigma_2^2\sigma_2\sigma_1\sigma_3\sigma_2^2\sigma_1$.

Signalons pour fixer les idées que le calcul d'une forme sommitale pour des mots de ce type nécéssite environ une heure de temps CPU sur un IBM 9370.

En conclusion, le problème de la classification, même à nombre de brins constant, reste difficile à traiter. Une voie d'amélioration qui semble prometteuse serait de regarder les groupes $B(n)$ à travers les représentations linéaires de Burau (cf [Bi] p 100) dès que l'obstacle de la question de la fidélité de celles-ci sera levé.

ANNEXE : le programme PROLOG

A l'origine, PROLOG (contraction de PROgrammer en LOGique) est adapté à décider si un prédicat (le *but*) est logiquement compatible avec un certain nombre de faits de base, et les faits qui s'en déduisent au moyen des règles d'inférence décrites par le programmeur. Mais on peut aussi *calculer* avec PROLOG : pour décrire une action $\mathcal{A}$ sur un objet source OS donnant comme résultat un objet final OF, il suffit de construire un prédicat logique A(_ , _) à deux paramètres tel que A(OS,OF) soit *vrai*. Dans l'écriture de A, on indiquera toutes les caractéristiques qui permettent d'identifier le deuxième argument comme résultat de l'action $\mathcal{A}$ sur le premier argument. Cet aspect *descriptif* de la programmation constitue un premier avantage de PROLOG, avantage renforcé par le fait que l'écriture reste très proche du langage mathématique. Un deuxième avantage est qu'il permet de décrire des actions qui peuvent donner à partir d'un même objet *plusieurs* résultats acceptables. Par exemple, dans un groupe de tresses l'action de simplification "triviale" $\sigma_i\sigma_i^{-1} \rightarrow e$ peut donner à partir du mot $\sigma_i\sigma_i^{-1}\sigma_j\sigma_j^{-1}$ les résultats acceptables $\sigma_i\sigma_i^{-1}$ ou $\sigma_j\sigma_j^{-1}$. Enfin, de par son mode de travail, le moteur d'inférence de PROLOG donnera toutes les solutions au problème posé, sans que le programmeur ait besoin d'évaluer à l'avance leur nombre et la façon de les ranger.

Dans ce qui suit, on donne la définition des formes normales de Garside et des formes sommitales dans l'univers PROLOG : pour la clarté, on a écrit les prédicats avec un formalisme simplifié par rapport au PROLOG standard. En particulier, certains paramètres "techniques" destinés à optimiser la vitesse d'exécution ont été omis, et on n'a pas détaillé certains prédicats intermédiaires (il y en a environ soixante dans le programme source). Le moteur d'inférence de PROLOG évalue chaque prédicat dans l'ordre d'écriture. En cas de succès, il continue avec le prédicat suivant, en cas d'échec, il remonte au prédicat antérieur, cherche s'il existe une autre solution le satisfaisant et si oui recommence avec cette nouvelle solution. On peut contrôler ce mode de résolution avec deux prédicats prédéfinis: le prédicat **coupe**, qui empêche l'essai d'une autre solution d'un prédicat qui

lui est antérieur, et le prédicat **échec** toujours faux et qui donc provoque la recherche d'autres solutions des prédicats antérieurs. Dans tout ce qui suit, le symbole _ désigne la variable anonyme, instanciée à une valeur quelconque au moment de l'évaluation.

Précisons le modèle choisi pour représenter les tresses: une écriture d'un mot de tresse sera considérée par le programme comme une *liste* d'entiers relatifs: au générateur σ_i on associe l'entier i, à σ_i^{-1} on associe -i. Ainsi par exemple le mot $\sigma_1\sigma_2^3\sigma_3^{-1}$ sera représenté par la liste **1 . 2 . 2 . 2 . -3 . vide**

La multiplication des mots se traduit naturellement par la concaténation des listes, et cette concaténation entre deux listes se fera via le prédicat à trois paramètres concatene(_,_,_).

L'extraction.

L'extraction utilise les facultés de récursivité de PROLOG. On va distinguer trois prédicats :

migre_g(entier,liste,liste) : fait migrer l'entier à gauche de la première liste, donnant comme résultat la seconde.

extraction_g(entier,liste,liste) : est vrai si l'entier s'extrait à gauche de la première liste, donnant comme reste la seconde.

extraction_de_mot(liste,liste,liste) : est vrai si la premiere liste s'extrait à gauche de la deuxième, donnant comme reste la troisième.

Voici maintenant les clauses nécessaires à leurs définitions:

migre_g(I , Liste_source , Liste_résultat) ←
concatene(Liste_début , J . I . Liste_fin , Liste_source) **et**
non(membre(I , Liste_début)) **et**
plusgrand(abs(J - I) , 1) **et**
concatene(Liste_début , I . J . Liste_fin , Liste_intermédiaire) **et**
migre_g(I , Liste_intermédiaire , Liste_résultat) **et**
coupe .

extraction_g(I , Liste_source , Liste_résultat) ←
migre_g(I , Liste_source , I . Liste_résultat) **et**
coupe .

extraction_g(I , Liste_source , Liste_résultat) ←
migre_g(I , Liste_source , Liste_intermédiaire) **et**
extraction_g(I , Liste_intermédiaire , Liste_résultat) **et**
coupe .

extraction_g(I , Liste_source , Liste_résultat) ←
consécutif(I , J) **et**
concatene(Liste_début , J . I . Liste_fin , Liste_source) **et**
non(membre(I , Liste_début)) **et**
extraction_g(J , Liste_fin , Liste_fin_mod) **et**
concatene(Liste_début , I . J . I . Liste_fin_mod , Liste_intermédiaire) **et**
extraction_g(I , Liste_intermédiaire , Liste_résultat) .

extraction_de_mot(**vide** , Liste_source , Liste_source) ←
coupe .

extraction_de_mot(I . Reste , Liste_source , I . Liste_Resultat) ←
extraction_g(I , Liste_source , Liste_intermédiaire) **et**
extraction_de_mot(Reste , Liste_intermédiaire , Liste_résultat) .

Forme de Garside.

On dispose ici des prédicats élémentaires
calcule_index(liste,entier) : vrai si l'entier coïncide avec le nombre de brins correspondant à la liste.
construit_delta(entier,liste) : calcule le mot Δ .
construit_inverses(entier) : construit tous les τ_i pour les $i < entier$.
élimine_neg(liste,liste) : réalise l'opération de la phase 2 de l'algorithme de Garside.
minimum_lexico(liste,liste) : vrai si la deuxième liste réalise le minimum suivant l'ordre lexicographique de la première.

Garside(Liste_source , Puissance , Reste) ←
calcule_index(Liste_source , Index) et
construit_delta(Index , Delta) et
construit_inverses(Index) et
élimine_neg(Liste_source , Nbre_delta , Liste_positive) et
extraction_de_mot(Delta , Liste_positive , Puissance , Liste_résiduelle) et
minimum_lexico(Liste_résiduelle , Reste) .

Forme sommitale.

Pour stocker les listes des ensembles S_i on utilise la faculté qu'a PROLOG d'enlever et d'ajouter des faits de manière dynamique à l'espace de travail. En pratique il faut construire un prédicat Existe(liste) et si l'on veut stocker la liste L on ajoutera le fait Existe(L) au moyen de l'opérateur **ajoute_fait** . L'opération inverse se réalise avec l'opérateur **retranche_fait** . De plus, ici, on stocke les listes sous formes de Garside, et on ne construit de nouvelles listes qu'à partir de celles qui n'ont pas déjà été utilisées. Il faut donc définir un prédicat à trois paramètres Existe(entier,liste,booléen), le booléen étant **vrai** si le couple (entier,liste) a été utilisé, **faux** sinon. La construction des formes initiales de Δ_n se fait de manière analogue, et ces formes initiales sont stockées à l'aide d'un prédicat Existe_Forme_Initiale(liste,booléen).

construit_sommit(Liste_source) ←
Garside(Liste_source , Puissance , Reste) et
ajoute_fait(Existe(Puissance , Reste , **faux**)) et
Existe(Puissance , Reste , **faux**) et
retranche_fait(Existe(Puissance , Reste , **faux**)) et
ajoute_fait(Existe (Puissance , Reste , **vrai**))[2] et
Existe_Forme_Initiale(F , _) et
conjugue(Puissance , Reste , F , Liste)[3] et
Garside(Liste , Puissance_nouvelle , Reste_nouveau) et
non(Existe(Puissance_nouvelle , Reste_nouveau , _)) et

[2] **vrai** sert à marquer la liste traitée
[3] réalise l'étape 2.1 ou 2.2 de l'algorithme

ajoute_fait(Existe_Forme_Initiale(Liste_transformée , **faux**)) et

plusgrand(Puissance_nouvelle , Puissance)[4] et

retranche_fait(Existe(_ , _ , _))[5] et

ajoute_fait(Existe(Puissance_nouvelle , Reste_nouveau , **faux**) et

échec .[6]

construit_sommit(_) .[7]

[4] teste si la puissance de Garside a augmenté, et sinon retour au point de départ

[5] on élimine toutes les listes de puissance désormais trop petite

[6] provoque le retour au point de départ

[7] clause triviale, pour terminer sur un succès quand toutes les listes ont été essayées

Bibliographie

[Bi] J. S. BIRMAN Braids, links and mapping class groups.

Princeton University Press.

[BZ] G. BURDE / H. ZIESHANG Knots. De Gruyter Studies in Mathematics.

[Ga] F. A. GARSIDE The braid group and other groups.

Quart. J. Math. Oxford 20 ; nbr 78 .

[Jo] V. F. R. JONES Hecke algebra representations of braid groups and link polynomials.

Annals of Math. nbr 126 ; 335-388

[Me] M. MORE Mémoire de programmation ENS Lyon / Lab. Topologie Dijon

[Mo] H. R. MORTON An irreducible 4 string braid with unknotted closure.

Math. Proc. Camb. Phil. Soc. nbr 93 ; 259-261

[Ro] D. ROLFSEN Knots and links. Publish or Perish inc.

[Ya] S. YAMADA The minimal number of Seifert circles equals the braid index of a link.

Inventiones Math. nbr 89 ; 347-356