TIPE – Tresses

C. Aubonnet, Y.-S. Chesnel-Bicep,
G. Dousson, A. Renout.

Géométrie, algèbre, topologie.

Résumé

Ce document est un résumé de travail autour de la notion de tresse. Il est séparé en une première partie mathématique qui s'inspire de l'idée commune de tresse puis une seconde informatique qui résout un problème dû à la représentation mathématique de ces tresses.

Dans un souci de simplicité, une approche géométrique assez informelle permet de se donner une première intuition du concept mathématique avant d'entamer une approche algébrique, plus riche mais plus complexe. Les preuves sont absentes et le formalisme souvent en note de bas de page pour éviter un bourbakisme inutile dans ce cadre proche de la vulgarisation.

Un algorithme est ensuite présenté, d'abord dans son principe théorique puis son implémentation pratique en OCaml. S'en suit une étude expérimentale de sa complexité moyenne, trop difficile à déterminer mathématiquement du fait de la diversité des tresses.

Table des matières

Ce document a également été rendu possible par l'aide de plusieurs contributeurs. Sont ainsi remerciés, entre autres ;

— B. Darnault qui a grandement aidé à approcher par des formules explicites les complexités de l'algorithme de retournement ;

— O. Laval-Sebire qui a participé à la création des graphiques de complexité.

1 Mathématique théorique.

1.1 Approche géométrique.

On s'intéresse aux tresses, vues comme un ensemble de brins (ficelles, cordes…) entremêlés dans la longueur. On définit donc naturellement les brins comme des courbes continues dans l'espace, à l'aide de fonctions, puis les tresses comme la réunion de ces brins, à condition qu'ils ne se coupent pas et qu'ils soient bien ancrés [1].

Définition 1.1 (Tresse géométrique)**.** On appelle tresse géométrique à n brins une famille de n fonctions continues de $[0,1]$ dans le disque unité ouvert de $\mathbb{C}$ telle que $\forall i,j \in [\![1,n]\!], \forall t \in [0,1], f_i(t) = f_j(t) \implies i = j$ et $\{f_1(0),\dots,f_n(0)\} = \{f_1(1),\dots,f_n(1)\}$.

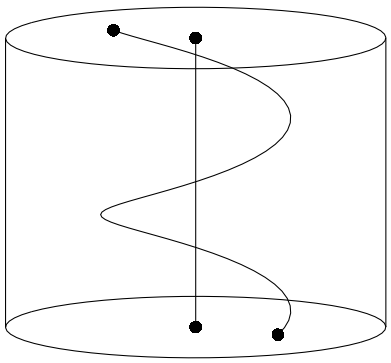

FIGURE 1 – Exemple de graphe d'une tresse géométrique à deux brins.

Remarque. On note $\mathfrak{B}_n$ l'ensemble des tresses géométriques à n brins.

On a donc désormais une définition mathématique de l'idée qu'on se fait d'une tresse. Cette définition seule est cependant stérile. On peut alors chercher des lois de composition et des relations entre les tresses géométriques. En restant proche de l'idée concrète de départ, une loi et une relation apparaissent naturellement.

1. On entend par là que chaque tresse a ses brins fixés aux mêmes points en "haut" et en "bas", de sorte qu'on pourra notamment les mettre bout à bout.

Définition 1.2 (Composition)**.** On définit la composition de deux tresses géométriques α et β, noté $\alpha \circ \beta$, comme la concaténation de ces deux tresses [2].

Définition 1.3 (Isotopie)**.** On dit que deux tresses géométriques α et β sont isotopes, noté $\alpha \sim \beta$, si on peut passer de l'une à l'autre continûment sans détacher les ancrages [3].

Théorème 1.1.1. *La relation d'isotopie est une relation d'équivalence sur* $\mathfrak{B}_n$*.*

Cette propriété tend à vouloir se détacher de l'idée géométrique d'une tresse vers une idée plus "topologique" où on considérerait qu'une tresse dont on ferait glisser un brin sans toucher à ses extrémités est toujours la même. Cela nous amène à une nouvelle définition de ce qu'on appellera tresse, en opposition à tresse géométrique.

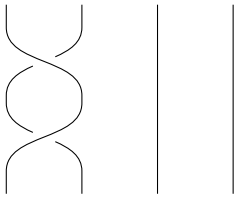

FIGURE 2 – Exemple de deux tresses géométriquement différentes et pourtant intuitivement similaires.

Définition 1.4 (Tresse)**.** On appelle tresse à n brins toute classe d'équivalence d'une tresse géométrique à n brins selon l'isotopie.

Remarque. On note $\mathcal{B}_n$ l'ensemble des tresses à n brins, c'est-à-dire $\mathcal{B}_n = \dfrac{\mathfrak{B}_n}{\sim}$.

Théorème 1.1.2. *La composition ne dépend pas des représentants choisis, et est donc bien définie sur* $\mathcal{B}_n$*.*

Démonstration. Informellement, en prenant α et α' deux tresses géométriques isotopes et β et β' de la même manière, on peut toujours, après concaténation, déformer α en α' et de même pour β. On vérifie bien $\alpha \circ \beta \sim \alpha' \circ \beta'$. □

Corollaire 1.1.1. $(\mathcal{B}_n, \circ)$ *est un groupe* [4]*.*

2. Formellement, on pourrait définir chaque brin h de la tresse résultante γ par morceaux avec un brin f de la tresse α et un brin g de la tresse β comme, pour $0 \leq t \leq \frac{1}{2}$, $h(t) = f(2t)$ et, pour $\frac{1}{2} \leq t \leq 1$, $h(t) = g(2t-1)$. On vérifierait alors bien que chaque brin h est continu, n'en coupe aucun autre et est bien ancré.

3. Formellement, α et β sont dites isotopes si il existe ϕ de $[0,1]$ dans $\mathfrak{B}_n$ continue et telle que $\phi(0) = \alpha$ et $\phi(1) = \beta$. La continuité de ϕ signifie ici que pour un x donné, lorsque t parcourt $[0,1]$, pour un brin f_t de $\phi(t)$ donné, l'application ψ définie par $\psi(t) = f_t(x)$ est continue.

4. En général non-abélien.

Démonstration. En se libérant des contraintes géométriques, la tresse sans croisement, qu'on appellera tresse vide, devient un élément neutre évident. Par ailleurs l'associativité est tout aussi naturelle ; mettre bout à bout deux tresses puis une troisième revient au même que mettre bout à bout une première tresse et la concaténation de deux autres. Enfin, le miroir d'une tresse est un symétrique car, en les composant, chaque croisement se simplifiera un à un. □

En partant de l'intuition, la définition initialement géométrique s'est mue en une définition plus proche de l'algèbre. Est donc justifiée une approche directement algébrique des tresses, soutenue par l'intuition que l'approche géométrique a bâtie. On ne s'attardera cependant pas sur la démonstration de l'équivalence [5] entre ces deux approches, preuve très dure par E. Artinen 1925[1] .

5. Pour être précis, il s'agira de montrer que les deux approches engendrent deux groupes qui sont isomorphes.

1.2 Approche algébrique.

Dans notre ancienne définition des tresses, il était assez clair que les croisements étaient centraux pour décrire les tresses. On pourra assimiler chaque croisement du k-ième brin par dessus le $k+1$-ième à une tresse élémentaire σ_k qui servira de générateur pour $\mathcal{B}_n$.

À ces générateurs, s'ajoute évidemment des relations. La première consiste à dire qu'un croisement entre deux brins peut être glisser indépendamment de ceux des autres brins. La seconde est, quant à elle, le glissement horizontal d'un grand croisement entre trois brins. La difficulté est de montrer que ces deux relations suffisent, ce qu'on admet ici.

Définition 1.5. On définit le groupe des tresses $\mathcal{B}_n$ par les générateurs σ_1, ..., σ_{n-1} et deux relations ;

— $\forall i, j \in [\![1, n]\!], |i-j| = 1 \implies \sigma_i\sigma_j\sigma_i \equiv \sigma_j\sigma_i\sigma_j$;

— $\forall i, j \in [\![1, n]\!], |i-j| > 1 \implies \sigma_i\sigma_j \equiv \sigma_j\sigma_i$.

On a donc une représentation algébrique des tresses. Les générateurs servent de lettres qui, ensemble, forment des mots capables de décrire n'importe quelle tresse. Ce formalisme rend bien plus facile l'étude que l'on peut faire des tresses mais introduit un problème qui ne s'était jusqu'alors pas poser, celui du mot.

Définition 1.6 (Équivalence)**.** Deux mots de tresse à n brins u et v sont dits équivalents, noté $u \equiv v$, si ils représentent la même tresse, c'est-à-dire si on peut passer de l'un à l'autre avec les relations de $\mathcal{B}_n$.

Définition 1.7 (Problème du mot)**.** On appelle problème du mot, étant donné deux mots de tresses, la question de leur équivalence.

Il est possible de résoudre algorithmiquement le problème du mot dans $\mathcal{B}_n$, ce qui n'est pas évident *a priori*. En effet, en 1953, A. Tarski a donné une preuve de l'impossibilité de résoudre le problème du mot dans le cas général[2]. Cette question sera traitée par la suite. On se propose auparavant de profiter de la richesse de l'approche algébrique pour relever quelques curiosités, qui pourront avoir leur intérêt par la suite.

Déjà, on peut s'intéresser aux morphismes de $\mathcal{B}_n$. On pouvait déjà pressentir ce qui devient évident avec notre nouvelle définition, à savoir l'inclusion, pour tout n, de $\mathcal{B}_n$ dans $\mathcal{B}_{n+1}$. En effet, l'injection canonique est un morphisme injectif évident de $\mathcal{B}_n$ dans $\mathcal{B}_{n+1}$. Par ailleurs, un autre morphisme très différent peut nous permettre de tirer un sous-ensemble de tresses présentant de bonnes propriétés, les tresses pures.

Définition 1.8. On pose une application pour laquelle chaque générateur σ_k a pour image la permutation des accroches associée. Cette application respecte évidemment la composition. Il s'agit donc d'un morphisme de $\mathcal{B}_n$ vers l'ensemble des permutations de $[\![1, n]\!]$.

On appelle alors ensemble des tresses pures $\mathcal{P}_n$ le noyau de cette application, c'est-à-dire l'ensemble des tresses qui ne permutent pas leurs accroches.

Théorème 1.2.1. *$\mathcal{P}_n$ est un sous-groupe de $(\mathcal{B}_n, \circ)$.*

Ce sous-ensemble s'avérera utile par la suite. Il existe de nombreux autres morphismes pertinents liant $\mathcal{B}_n$ à d'autres groupes usuels, il est notamment à souligner que $\mathcal{B}_2$ est isomorphe à $\mathbb{Z}$ [6]. On propose cependant de s'intéresser à une dernière propriété inattendue de $\mathcal{B}_n$, qui mettra d'autant plus en lumière la richesse du groupe des tresses.

Théorème 1.2.2. *Il est possible d'engendrer $\mathcal{B}_n$ avec deux générateurs et deux relations.*

Démonstration. □

6. On pourra par exemple remarquer que tout élément de $\mathcal{B}_2$ est de la forme σ_1^k avec $k \in \mathbb{Z}$.

2 Algorithme de retournement.

2.1 Principe.

Pour résoudre le problème du mot, on va essayer de le simplifier. Pour commencer, on le réduit à la question de l'équivalence d'un mot avec le mot nul ϵ. Pour continuer, on peut utiliser les relations du groupe des tresses pour réarranger les mots en des formes plus simples. Par exemple, une forme naturelle consisterait à avoir tous les générateurs à gauche et tous les inverses à droite.

Pour cela, on définit une relation de réécriture en ne conservant que les cas particuliers des relations du groupe allant dans le sens de ce tri. Une fois un système de réécriture établi, on pourra étudier ses propriétés qui devraient aider à résoudre le problème du mot. On vérifiera avant tout que ce système de réécriture converge pour s'assurer qu'il soit bien utilisable.

Théorème 2.1.1. *Déterminer si un mot représente la tresse vide revient à résoudre le problème du mot.*

Démonstration. Soit w_1 et w_2 deux mots. Résoudre le problème du mot revient à savoir si w_1 et w_2 représente la même tresse, c'est-à-dire si $w_1 w_2^{-1}$ représente la tresse vide. □

Définition 2.1 (Réécriture directe $\to$)**.**

$$\forall u, v \in \mathcal{B}_n, \begin{cases} \forall i & & u\sigma_i^{-1}\sigma_i v & \to & uv \\ |i-j|=1 & \implies & u\sigma_i^{-1}\sigma_j v & \to & u\sigma_j\sigma_i\sigma_j^{-1}\sigma_i^{-1}v \\ |i-j|>1 & \implies & u\sigma_i^{-1}\sigma_j v & \to & u\sigma_j\sigma_i^{-1}v \end{cases}$$

Définition 2.2 (Réécriture $\to^\star$)**.** On définit la réécriture $\to^\star$ comme la clôture transitive de la réécriture directe $\to$, c'est-à-dire, pour deux mots $u, v \in \mathcal{B}_n$, $u \to^\star v$ si et seulement si il existe une suite finie[7] de mots (w_k) telle que $w_0 = u$, $w_n = v$ et, pour tout $k \in [\![0, n-1]\!]$, $w_k \to w_{k+1}$.

Remarque. On vérifie par une récurrence immédiate[8] que pour deux mots u et v de $\mathcal{B}_n$ tels que $u \to^\star v$, on a $u \equiv v$, de sorte qu'en réécrivant un mot donné on soit toujours en mesure de conclure sur la tresse qu'il représente.

Lemme 2.1.1 (Convergence)**.** *Toute suite (w_k) de mots de $\mathcal{B}_n$, telle que, pour tout $k \in \mathbb{N}$, on vérifie $w_k \to w_{k+1}$, est stationnaire.*

7. On entend par là une famille finie indexée par $[\![0, n]\!]$ avec $n \in \mathbb{N}$.

8. Étant donné que la réécriture directe $\to$ se base sur des cas particuliers des relations du groupe des tresses et que l'équivalence des mots $\equiv$ est transitive.

Lemme 2.1.2 (Confluence)**.** *Si pour un mot u de $\mathcal{B}_n$ donné, il existe deux mots v et v' de $\mathcal{B}_n$ tels que $u \to^\star v$ et $u \to^\star v'$, alors il existe un mot w de $\mathcal{B}_n$ tel que $v \to^\star w$ et $v' \to^\star w$.*

Théorème 2.1.2 (De réécriture positive)**.** *Pour tout mot $w \in \mathcal{B}_n$, il existe deux uniques mots positifs*[9] *$u, v \in \mathcal{B}_n$ tels que $w \to^\star uv^{-1}$.*

Démonstration. L'existence est garantie par le lemme de convergence et l'unicité par le lemme de confluence. □

Le système de réécriture posé converge donc bien vers une forme plus simple de mots sur laquelle on peut se concentrer. Il ne reste dès lors plus qu'à donner une condition nécessaire et suffisante sur la réécriture positive qui garantirait l'équivalence avec le représentant de la tresse vide ϵ.

Corollaire 2.1.1 (De réécriture vide)**.** *Pour deux mots de tresses positifs u et v, $uv^{-1} \equiv \epsilon$ si et seulement si $u^{-1}v \to^\star \epsilon$.*

On peut donc résoudre le problème du mot en écrivant un algorithme qui réécrirait un mot donné en sa forme positive. Un tel algorithme de réécriture est appelé algorithme de retournement. Il est à noter que, pour se faire, cet algorithme devrait être utilisé deux fois.

9. Un mot positif est un mot qui ne s'écrit qu'avec des générateurs et aucun inverse.

2.2 Implémentation.

On se propose d'implémenter l'algorithme de retournement en OCaml pour en faire une étude expérimentale par la suite.

On commence par définir les lettres, appelées sigmas, comme des entiers relatifs. Un entier strictement positif k représente σ_k, un entier strictement négatif k représente σ_{-k}^{-1} et 0 représente ϵ. Un mot de tresse n'est alors plus qu'une liste de sigmas.

Pour l'algèbre de base, la composition est déjà implémentée en tant que concaténation de deux listes. Il faut cependant encore ajouter l'inversion qui consiste en un miroir de la liste et la symétrisation de tous les sigmas, c'est-à-dire leurs remplacements par leurs opposés.

On est alors prêt à programmer l'algorithme de retournement. Pour se faire, on écrit une sous-fonction correspondant à une réécriture directe [10] puis on l'applique tant qu'elle change quelque chose, c'est-à-dire tant que la suite de mots réécrits ne stationne pas. Une fonction permet alors d'identifier les deux mots positifs dans le résultat réécrit.

Il ne reste plus qu'à appliquer l'algorithme de retournement ainsi construit pour tester si un mot représente la tresse vide puis, plus généralement, pour résoudre le problème du mot.

Listing 1 – Structures de données

```
type sigma = int ;;
type word = sigma list ;;
```

Listing 2 – Fonction d'inversion

```
let inverse (w :word) :word =
    let rec aux w acc = match w with
        [] -> acc
        |t ::q -> aux q (-t ::acc)
    in aux w [];;
```

10. On introduit cependant une légère différence. En effet, on simplifie les lettres même lorsque l'inverse est à gauche, ce qui n'est pas le cas dans la réécriture directe définie plus haut. Les résultats de convergence et confluence restent les mêmes mais en permettant de réduire un peu plus le nombre de lettres, on réduit sensiblement la complexité.

Listing 3 – Fonction de réécriture

```
let rec rewrite (w :word) =
    let rec step w = match w with
        []
            -> [], false
        |0 ::q
            -> let s = step q in fst s, true
        |t1 ::t2 ::q when t1 = -t2
            -> let s = step q in fst s, true
        |t1 ::t2 ::q when t1 < 0 && t2 > 0 && abs(t1+t2) = 1
            -> let s = step q in t2 ::(-t1) ::(-t2) ::t1 ::(fst s), true
        |t1 ::t2 ::q when t1 < 0 && t2 > 0 && abs(t1+t2) > 1
            -> let s = step q in t2 ::t1 ::(fst s), true
        |t ::q
            -> let s = step q in t ::(fst s), snd s
    in let w, c = step w
    in if c then rewrite w else w;;
```

Listing 4 – Fonction d'identification

```
let break_down (w :word) :word*word =
    let rec aux w u v = match w with
        [] -> (inverse u), v
        |t ::q when t>0 -> aux q ((-t) ::u) v
        |t ::q -> aux q u ((-t) ::v)
    in aux w [] [];;
```

Listing 5 – Résolution du problème du mot

```
let is_empty (w :word) =
    let u,v = break_down (rewrite w) in rewrite ((inverse u)@v) = [];;

let is_equiv (w1 :word) (w2 :word) = is_empty (w1@(inverse w2));;
```

2.3 Étude expérimentale.

On peut désormais faire une étude expérimentale de la résolution du problème du mot en utilisant l'algorithme de retournement, et plus précisément sur la fonction is_empty [11]. Pour se faire, on utilise le module Sys de OCaml qui permet de mesurer le temps processeur depuis le début de l'exécution. Plusieurs facteurs vont cependant jouer sur l'incertitude et altérer les mesures.

Pour pouvoir reproduire ces expériences, les résultats seront toujours accompagnés des trois paramètres utilisés ;

- l correspondant au nombre de lettres, c'est-à-dire la longueur de la liste entrée en paramètre ;
- n correspondant au nombre de brins, c'est-à-dire le maximum [12], en valeur absolue, des entiers de la liste ;
- N correspondant au nombre d'appels, sur des tresses différentes, généralement sous la forme d'une fonction de l.

Toutes les tâches parallèles (WiFi, sauvegardes, autres programmes…) ont été réduites pendant les mesures [13]. Le paramètre N a été maximisé pour une représentativité optimale des échantillons et une meilleure précision des mesures dans un compromis de temps de calcul raisonnable, pouvant rapidement atteindre plusieurs heures.

Une première étude sur les deux paramètres simultanément permet d'avoir une vue d'ensemble de la complexité.

Pour comprendre la forme de la complexité en fonction du nombre de brins, on peut répertorier dans un tableau l'impact de chaque relation de réécriture directe [14] et sa probabilité d'apparition pour de lettres voisines données en fonction de n.

On constate que la dernière relation, sans impact sur la longueur, tend à s'imposer ce qui explique la valeur constante. Pour les premiers brins les autres relations sont encore très présentes et impactent dans des sens très différents ce qui explique la forme en cloche, pouvant faire penser à une loi normale. Il est cependant à souligner que la réécriture n'étant pas aléatoire,

11. Bien que ce soit la fonction is_equiv, qui résout le problème du mot, c'est bien la fonction is_empty donc la complexité est indéterminée.

12. Pour être exact, ce maximum est $n-1$ étant donné que σ_k correspond au croisement du brin k et $k+1$. Il ne s'agit cependant que d'une translation dans les graphes qui n'aura aucun impact sur la forme.

13. Bien qu'on mesure le temps processeur, d'autres tâches pourraient avoir des effets de bord, par exemple en appelant plus fréquemment le *garbage collector*.

14. La relation $u\sigma_i\sigma_i^{-1}v \to uv$ englobera la simplification des inverses en général, même de la forme $u\sigma_i^{-1}\sigma_i v$, puisque l'algorithme l'implémente.

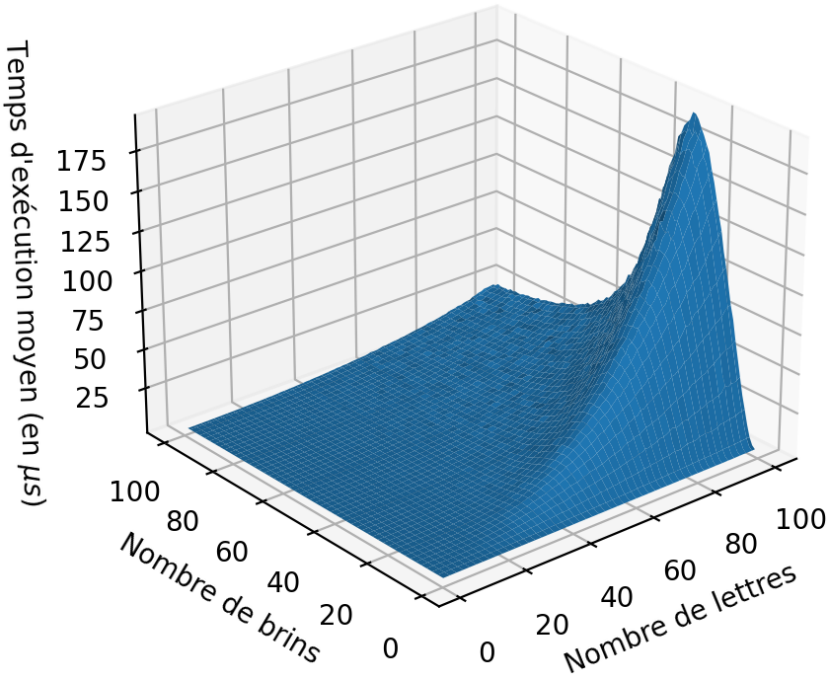

FIGURE 3 – Expérience réalisée avec l et n variant de 1 à 100 et, à chaque étape, $N = 10^6/l$.

Relation	Probabilité	Longueur
$u\epsilon v \to uv$	$\frac{1}{2n-1}$	-1
$u\sigma_i\sigma_i^{-1}v \to uv$	$\frac{2(n-1)}{(2n-1)^2}$	-2
$u\sigma_i^{-1}\sigma_j v \to u\sigma_j\sigma_i\sigma_j^{-1}\sigma_i^{-1}v$	$\frac{2(n-2)}{(2n-1)^2}$	$+2$
$u\sigma_i^{-1}\sigma_j v \to u\sigma_j\sigma_i^{-1}v$	$\frac{(n-3)(n-2)+2(n-4)}{(2n-1)^2}$	0

FIGURE 4 – Probabilité d'existence d'une relation de réécriture directe entre deux lettres d'un mot à n brins et son impact sur la longueur du mot.

la probabilité d'apparition de ces relations changent avec le temps [15] ce qui empêche de déterminer facilement une formule explicite de la complexité selon le nombre de brins.

On comprend cependant que la complexité selon le nombre de brins est en réalité dû à la complexité selon la longueur du mot et que le nombre de brins ne fait qu'impacter cette longueur. La complexité en temps est donc proportionnelle au produit de la complexité selon la longueur par l'impact

15. Par exemple, la relation augmentant la longueur fait apparaître des inverses qui peuvent très vite se simplifier si on a encore peu de brins, donc peu de lettres différentes.

moyen sur la longueur (c'est-à-dire la "complexité" selon le nombre de brins), ce qui simplifie notre étude.

Annexe.

Quelques améliorations.

On peut poser une application qui, à chaque générateur de $\mathcal{B}_n$, associe la permutation entre k et $k+1$. On construit ainsi un morphisme de $\mathcal{B}_n$ dans l'ensemble des permutations de $[\![1, n]\!]$. Tout mot du noyau de ce morphisme est alors représentant d'une tresse dite pure. Il s'agit d'une condition nécessaire [16] pour être vide.

On peut alors faire un algorithme testant la pureté d'une tresse en complexité linéaire, et, la majorité des tresses n'étant pas pures, permettre un premier filtrage avant d'appliquer l'algorithme de retournement, plus coûteux.

Le problème du mot sur $\mathcal{B}_n$ est P.

Le problème de savoir si, pour un mot donné, il existe un mot plus court représentant la même tresse est NP-complet.

16. Pas suffisante cependant. Par exemple la tresse représentée par $\sigma_1\sigma_1$ n'est pas vide bien qu'elle ait pour image l'identité.

Références

[1] Emil Artin. Theory of braids. *Annals of Mathematics*, pages 101–126, 1947.

[2] Alfred Tarski, Andrzej Mostowski, and Raphael Mitchel Robinson. *Undecidable theories*, volume 13. Elsevier, 1953.