TIPE – Tresses

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

Géométrie, algèbre, topologie.

Abstract

The braid theory is linked to many fields of mathematics. Consequently, this document begins with intuitive geometry helped by analysis and calculus. Then, a more rigorous view of the subject is developed using algebra. However, proofs are missing and formalism left in footnotes to simplify reading.

This algebraic perspective leads to an issue: the word problem. Unsolvable in universal algebra, this very specific case allows us to work out an algorithm to resolve it. This algorithm is developed in the second part, with its implementation in OCaml along with an experimental study of it.

Table des matières

Ce document a également été rendu possible par l'aide de plusieurs personnes. Sont ainsi remercié·e·s, entre autres ;

— B. Beaujean, B. Darnault et G. Richomme, pour les formules approchées de complexité ;
— O. Laval-Sebire, pour les 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 $\{f_1(0),\dots,f_n(0)\} = \{f_1(1),\dots,f_n(1)\}$.

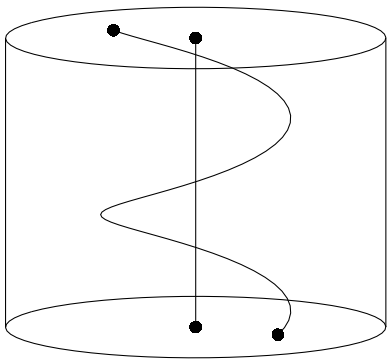

FIGURE 1 – Exemple de graphe d'une tresse géométrique à deux brins.

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.

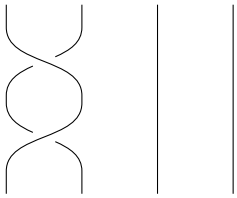

FIGURE 2 – Exemple de deux tresses géométriquement différentes et pourtant intuitivement similaires.

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. Artin en 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 glissé 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 ;

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

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

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 posé, 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] ce qui peut servir à établir une nouvelle propriété de $\mathcal{B}_n$.

Lemme 1.2.1. *Pour tout $n \in \mathbb{N}$, $\mathcal{B}_n \in \mathcal{B}_{n+1}$.*

Théorème 1.2.2. $\operatorname{card} B_1 = 1$ *et, pour tout $n \geq 2$,* $\operatorname{card} B_n = \aleph_0$.

Démonstration. $\mathcal{B}_1$ ne contient aucun générateur, il ne possède qu'un élément, ϵ.

$\mathcal{B}_2$ est isomorphe à $\mathbb{Z}$, en particulier il existe une bijection entre les deux. On a donc $\operatorname{card} \mathcal{B}_2 = \operatorname{card} \mathbb{Z} = \operatorname{card} \mathbb{N} = \aleph_0$.

Pour $n \geq 3$, on sait que $\mathcal{B}_2 \subset \mathcal{B}_n$. Ainsi $\operatorname{card} \mathcal{B}_n \geq \operatorname{card} \mathcal{B}_2 = \aleph_0$. De plus, tout groupe engendré par une partie finie est dénombrable, c'est-à-dire $\operatorname{card} \mathcal{B}_n \leq \aleph_0$, d'où $\operatorname{card} \mathcal{B}_n = \aleph_0$. □

Il existe de nombreuses autres curiosités à propos du groupe des tresses. On propose cependant de s'intéresser à une dernière, inattendue, qui mettra d'autant plus en lumière la richesse de la théorie.

Théorème 1.2.3. *Il est possible d'engendrer $\mathcal{B}_n$ avec deux générateurs et deux relations.*

Démonstration. On pose, pour le groupe des tresses $\mathcal{B}_n$, les générateurs $\tau = \sigma_{n-1} \dots \sigma_1$ et $\sigma = \sigma_1$ et deux relations ;

— $\sigma\tau\sigma\tau^{-1}\sigma\tau \equiv \tau\sigma\tau^{-1}\sigma\tau\sigma$;

— $\forall k \in [\![2, n-2]\!], \sigma\tau^k\sigma\tau^{-k} \equiv \tau^k\sigma\tau^{-k}\sigma$.

On vérifie aisément, avec la définition de $\mathcal{B}_n$, que, pour tout $i \in [\![0, n-2]\!]$, $\tau\sigma_{i+1} \equiv \sigma_i\tau$. Il en suit, par récurrence, que $\sigma_{i+1} \equiv \tau^{-i}\sigma\tau^i$, ce qui permet d'obtenir les deux relations sur τ et σ à partir des deux relations de définition de $\mathcal{B}_n$.

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

En partant de τ et σ, on peut obtenir les générateurs de définition de $\mathcal{B}_n$ avec la formule précédente. On vérifie alors réciproquement qu'on peut déduire la définition initiale de $\mathcal{B}_n$ à $n-1$ générateurs de cette caractérisation à deux générateurs. $\square$

2 Algorithme de retournement.

2.1 Principe.

De nombreux algorithmes ont été développés pour résoudre le problème du mot. A déjà été cité E. Artin qui en présentait déjà un dans ses travaux fondateurs[1], implémenté par la suite[3] mais trop coûteux pour être utilisable en pratique. P. Dehornoy en a à son tour proposé un[4], nettement plus recherché mais aussi plus rapide. On se propose un compromis entre efficacité et accessibilité avec l'algorithme de retournement, également étudié par P. Dehornoy dans une section d'un de ses derniers livres[5], auquel on pourra se référer pour les preuves subtiles éclipsées ici.

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.

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 $\to$)**.** Pour deux mots $u, v \in \mathcal{B}_n$,

$$\begin{cases} & u\sigma_i^{-1}\sigma_i v & \to & uv \\ |i-j|=1 & \implies & u\sigma_i^{-1}\sigma_j v & \to & u\sigma_j\sigma_i\sigma_j^{-1}\sigma_i^{-1}v \\ |i-j|>1 & \implies & u\sigma_i^{-1}\sigma_j v & \to & u\sigma_j\sigma_i^{-1}v \end{cases}$$

Définition 2.2 (Réécriture $\to^\star$)**.** Pour deux mots $u, v \in \mathcal{B}_n$, $u \to^\star v$ si et seulement si il existe une suite finie de mots $(w_k)_{k\in[\![0,n]\!]}$ telle que $w_0 = u$, $w_n = v$ et, pour tout $k \in [\![0, n-1]\!]$, $w_k \to w_{k+1}$, c'est-à-dire si on peut passer de u à v par un nombre fini de réécritures directes successives.

Remarque. On vérifie par une récurrence immédiate[7] 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.

7. 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.

Maintenant le système de réécriture établi, il s'agit d'étudier ses propriétés qui devraient aider à résoudre le problème du mot. On vérifie avant tout que ce système de réécriture converge pour s'assurer qu'il soit bien utilisable, ensuite que présente bien une forme unique.

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

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 (Existence et unicité de la décomposition d'Ore)**.** *Pour tout mot $w \in \mathcal{B}_n$, il existe deux uniques mots positifs*[8] *$u, v \in \mathcal{B}_n$ tels que $w \to^\star uv^{-1}$.*

Le système de réécriture posé converge 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 (Théorème d'Ore)**.** *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, la première pour obtenir l'écriture positive, la seconde pour vérifier ou non la condition.

8. 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 l'élément neutre ϵ. Un mot de tresse n'est alors jamais 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 qui parcourt le mot, stocke chaque lettre passée, applique une relation de réécriture directe dès que nécessaire puis en recommençant son parcours une lettre en arrière et finalement renvoie l'ensemble des lettres parcourue et réécrite. Elles sont cependant dans le mauvais sens, il s'agit donc de les retourner avec un "miroir" pour obtenir le mot réécrit.

On complète avec une fonction séparant les deux mots positifs d'un mot réécrit, en parcourant le premier jusqu'à s'arrêter à la séparation entre les deux. Il ne reste alors 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 [] ;;
```

Listing 3 – Fonction de réécriture

```
let rewrite w =
    let rec mirror w acc = match w with
        [] -> acc
        |t ::q -> mirror q (t ::acc)
    in
    let rec aux w1 w2 = match w1 with
        [] -> w2
        |0 ::q1 -> (match w2 with
            [] -> aux q1 []
            |h ::q2 -> aux (h ::q1) q2)
        |i ::j ::q1 when i<0 && j>0 && abs(i+j)=0 -> (match w2 with
            [] -> aux q1 []
            |h ::q2 -> aux (h ::q1) q2)
        |i ::j ::q1 when i<0 && j>0 && abs(i+j)=1 -> (match w2 with
            [] -> aux (j ::(-i) ::(-j) ::i ::q1) []
            |h ::q2 -> aux (h ::j ::(-i) ::(-j) ::i ::q1) q2)
        |i ::j ::q1 when i<0 && j>0 && abs(i+j)>1 -> (match w2 with
            [] -> aux (j ::i ::q1) []
            |h ::q2 -> aux (h ::j ::i ::q1) q2)
        |h ::q -> aux q (h ::w2)
    in mirror (aux w []) [];;
```

Listing 4 – Fonction d'identification

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

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.

Déterminer la complexité de cet algorithme est un problème en soi. On peut fragmenter ce calcul en déterminant la complexité de la fonction de réécriture, celle d'inversion, de concaténation et d'identification puis en faire la somme. En effet, la résolution du problème du mot par retournement consiste en une inversion, une concaténation, deux réécritures et une identification [9].

On constate facilement que la fonction d'inversion présente une complexité linéaire et la documentation OCaml renseigne sur celle de la concaténation, linéaire également. La fonction d'identification est moins évidente mais on se convainc assez rapidement qu'elle sera aussi en $\mathcal{O}(n)$. Toute la question réside donc dans la détermination de la complexité de la fonction de réécriture, qui sera le cas traité dans la suite de cette section.

Il est immédiat que la complexité "meilleur cas" est linéaire [10]. Pour autant, la majorité des appels consistent en l'application des différentes opérations de réécriture avec des retours en arrière dans la lecture du mot, augmentant considérablement le nombre d'opérations à faire. Au-delà de la longueur de la tresse, ses lettres et leur agencement comptent donc ce qui rend difficile de déterminer une complexité moyenne mathématiquement. En conséquence, on se propose de procéder à des mesures expérimentales du temps d'exécution moyen d'une réécriture.

Relation	Probabilité	Longueur
$u\epsilon v \to uv$	$\frac{1}{2n-1}$	-1
$u\sigma_i^{-1}\sigma_i v \to uv$	$\frac{2(n-1)}{(2n-1)^2}$	-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 3 – 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.

Avant cela, on peut tout de même souligner qu'à l'aide d'un peu de dénombrement, on établit facilement la probabilité de présence de chaque relation au milieu d'un mot de tresse aléatoire. Au fur et à mesure de la réécriture, les mots tendent cependant à s'organiser rendant insuffisants ces calculs. On remarquera tout de même que, le nombre de brins augmentant, la relation

9. On se place ici dans le cas de comparaison à un mot représentant la tresse vide, il faut ajouter un inverse et une concaténation supplémentaire dans le cas général.

10. Ce meilleur cas arrive lorsque l'appel de la fonction revient toujours au dernier cas, c'est-à-dire lire les lettres une par une jusqu'à la fin. Il arrive avec des mots déjà trié (sans lettre neutre).

"d'échange" ($u\sigma_i^{-1}\sigma_j v \to u\sigma_j\sigma_i^{-1}v$) tend à s'imposer tandis que les autres se raréfient, alors même qu'il s'agit de la seule relation sans impact sur la longueur du mot. On peut donc s'attendre à une complexité indépendante du nombre de brins si ce dernier est suffisamment grand.

Avec des mesures, on constate effectivement ce phénomène qui se traduit par une convergence du temps d'exécution vers une valeur constante à nombre de lettres fixé. On constate également un pic qui s'explique par le fait que la relation "de duplication" ($u\sigma_i^{-1}\sigma_j v \to u\sigma_j\sigma_i\sigma_j^{-1}\sigma_i^{-1}v$) est, au début, compensé par la relation de suppression ($u\sigma_i\sigma_i^{-1}v \to uv$) tant que le nombre de brins est faible, les lettres ajoutées ayant de grandes chances de disparaître rapidement. Le nombre de brins augmentant, les lettres ajoutées sont plus rares mais vont entraîner de plus en plus de relations non-simplificatrices ce qui explique l'augmentation, jusqu'à un maximum à partir duquel la raréfaction de la duplication n'est plus suffisamment compensé. On tend alors vers la valeur constante, la relation d'échange s'imposant alors.

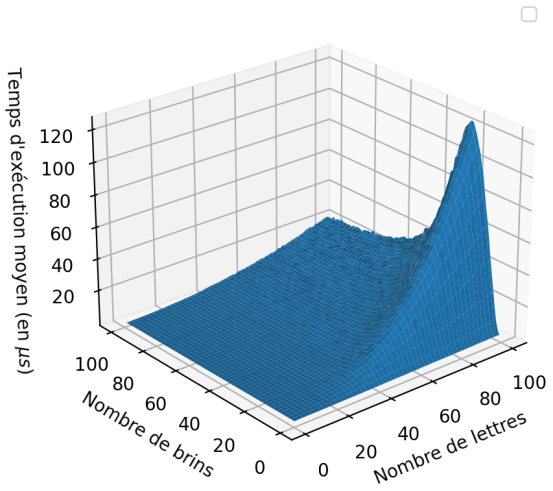

FIGURE 4 – Expérience réalisée avec un nombre de mots et de brins variant indépendamment de 1 à 100 et, à chaque étape, $10^6/\sqrt{l}$ mots aléatoires testés (avec l la longueur de celles-ci).

On s'intéresse malgré tout principalement à la complexité selon la longueur des mots, qui apparaît comme n'étant clairement pas linéaire sur le graphique. Une nouvelle série de tests permet une meilleur vision de celle-ci.

Pour améliorer cette complexité, on pourrait vérifier si le mot donné

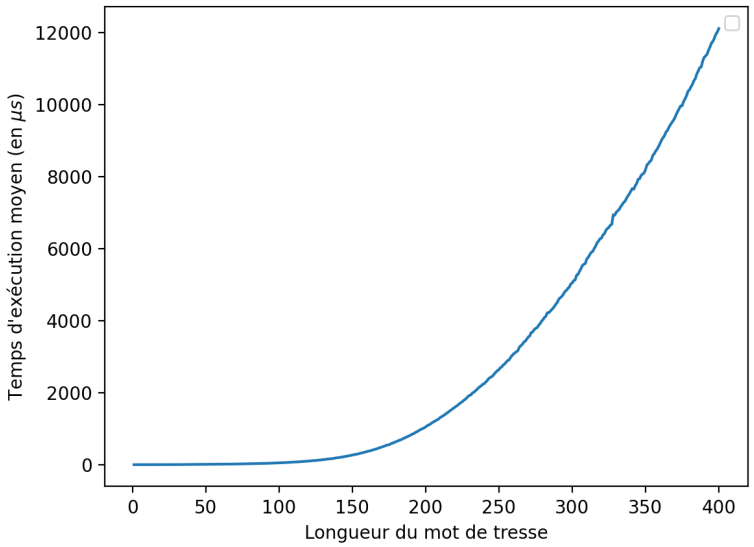

FIGURE 5 – Expérience réalisée avec un nombre de mots variant de 1 à 400 avec un nombre de brins fixé à 40 et, à chaque étape, $10^6/\sqrt{l}$ mots aléatoires testés (avec l la longueur de celles-ci).

représente une tresse pure, condition nécessaire pour être vide, un test en complexité linéaire qui permettrait un premier filtre avant d'appliquer un algorithme plus coûteux.

Listing 6 – Filtre par pureté

```
let is_pure w =
    let rec aux top bot n read_w w = match read_w with
        [] when bot = top -> if top < n then aux (top+1) (top+1) n w
            w else true
        |[] -> false
        |0 ::q -> aux top bot n q w
        |t ::q -> let x=abs t in if x=bot-1 then aux top (bot-1) n q w
                                 else if x=bot then aux top (bot+1) n q
                                     w
                                 else aux top bot n q w
    in aux 1 1 (max w) w w;;
```

Par ailleurs, sachant qu'on peut représenter B_n avec seulement deux générateurs à la place de n, on peut changer de structure de données pour coder les générateurs sur 2 bits et non plus en entiers de 63 bits (norme OCAML).

On divise de la sorte par $31,5$ la place en mémoire prise par un générateur. Il faut cependant vérifier que les listes, qui vont augmenter en longueur, ne dépasse pas ce facteur sans quoi la nouvelle structure de données prendrait plus de place que l'originale proposée.

Listing 7 – Structure de données avec deux générateurs

```
type letter = sigma list ;;
type word = letter list ;;
```

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.

[3] Elizabeth Luginbuhl. *The computer implementation of braid algorithms.* PhD thesis, University of Manitoba, 1973.

[4] Patrick Dehornoy. A fast method for comparing braids. *Advances in Mathematics*, 125(2) :200–235, 1997.

[5] Patrick Dehornoy. *Braids and self-distributivity*, volume 192. Birkhäuser, 2012.

[6] Patrick Dehornoy. Deux propriétés des groupes de tresses. *CR Acad. Sci. Paris*, 315 :633–638, 1992.

[7] Jae Choon Cha, Ki Hyoung Ko, Sang Jin Lee, Jae Woo Han, and Jung Hee Cheon. An efficient implementation of braid groups. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 144–156. Springer, 2001.

[8] Thomas Aubriot and Emmanuel Wagner. Des tresses et des nœuds en mathématiques. *L'Ouvert*, 113, 2006.

[9] Marc Autord. *Aspects algorithmiques du retournement de mot.* PhD thesis, Université de Caen, 2009.

[10] Juan González-Meneses. Basic results on braid groups. In *Annales Mathématiques Blaise Pascal*, volume 18, pages 15–59, 2011.

[11] Ester Dalvit. Braids : A movie for the popularisation of the mathematical theory of braids. *12th Conference on Applied Mathematics, APLIMAT 2013, Proceedings*, 01 2013.