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.

The OCaml module is available at https://github.com/gaspardousson/braid!

Table des matières

1 Mathématique théorique.

1.1 Approche géométrique.

On peut voir les tresses comme un ensemble de brins (ficelles, cordes…) entremêlés dans la longueur. Pour définir ces brins, on peut naturellement introduire l'idée de courbes continues dans l'espace, à l'aide de fonctions. On en tire immédiatement une définition des tresses comme réunion de 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 $\{f_1(0),\dots,f_n(0)\} = \{f_1(1),\dots,f_n(1)\}$.

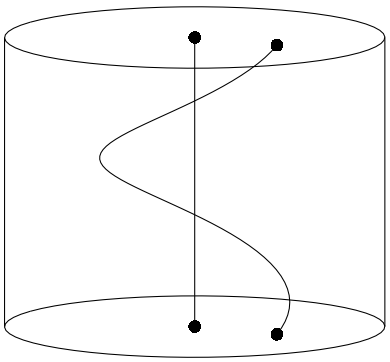

FIGURE 1 – Exemple de tresse géométrique à deux brins.

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

Cette définition mathématique de l'idée qu'on se fait d'une tresse est cependant stérile. On peut la compléter de lois de composition et de relations. En restant proche de l'idée concrète de départ, une loi et une relation semblent s'imposer.

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].

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 les mettre bout à bout.

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 de sorte que, 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".

Définition 1.3 (Isotopie)**.** On dit que deux tresses géométriques α et β sont isotopes, noté $\alpha \sim \beta$, si on peut déformer α continûment en β 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 abstraite 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.

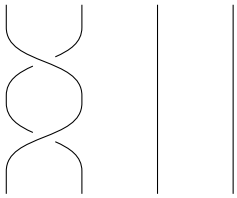

FIGURE 2 – Exemple de deux tresses géométriquement différentes et pourtant intuitivement similaires.

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 l'ensemble $\mathfrak{B}_n$ quotienté par la relation d'isotopie $\sim$.

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 de part et d'autre de la tresse résultante de la composition. On vérifie bien $\alpha \circ \beta \sim \alpha' \circ \beta'$. □

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

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 (pour $n \geq 3$).

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 la première tresse et la concaténation des 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 avérée voisine de l'algèbre. Est donc justifiée une approche directement algébrique des tresses, soutenue par l'intuition que l'approche géométrique a bâtie.

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 est une semi-commutativité en cela 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 à résumer toutes les autres, ce qu'on admet ici.

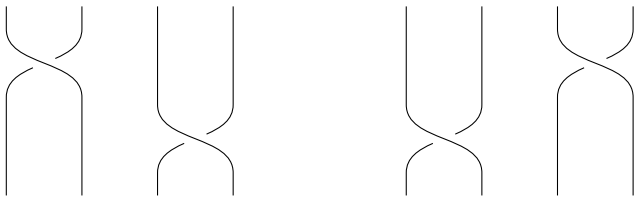

FIGURE 3 – Représentation de la relation de semi-commutativité des croisements.

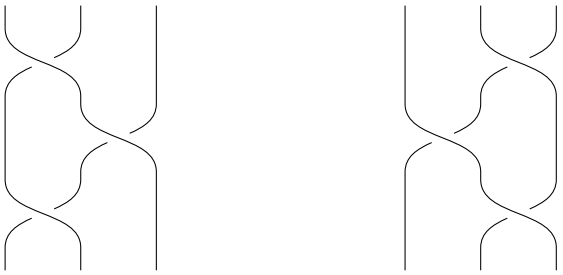

FIGURE 4 – Représentation de la relation d'échange horizontal des croisements.

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

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

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

Théorème 1.2.1. *Cette nouvelle définition de $\mathcal{B}_n$ est isomorphe à la précédente.*

Démonstration. En identifiant, comme déjà vu, les générateurs $\sigma_1, \ldots, \sigma_{n-1}$ à des croisements, E. Artin a pu démontrer en 1925 l'équivalence entre cette nouvelle approche algébrique et la précédente géométrique. □

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 (Problème du mot)**.** On appelle problème du mot, étant donné deux tresses, la question de leur isotopie [5].

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 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 1.7. 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 $\mathfrak{S}_n$ [6].

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. En tant que noyau, il s'agit donc d'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 $\mathbb{Z}$ [7] ce qui peut servir à établir une nouvelle propriété de $\mathcal{B}_n$.

5. La question est donc de déterminer si on peut déformer une tresse en une autre dans le cas général.

6. La notation $\mathfrak{S}_n$ désigne le groupe symétrique d'indice n, c'est-à-dire l'ensemble des permutations de $[\![1, n]\!]$

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

Théorème 1.2.2. $\operatorname{card} B_0 = \operatorname{card} B_1 = 1$ *et, pour tout* $n \geq 2$, $\operatorname{card} B_n = \aleph_0$.

Démonstration. $\mathcal{B}_0$ et $\mathcal{B}_1$ ne contient aucun générateur, il ne possède qu'un élément, ϵ. Autrement dit, on ne peut pas faire de tresses avec un brin ou aucun.

Par ailleurs, $\mathcal{B}_2$ 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. On en déduira des conséquences dans d'autres domaines mathématiques avant de se focaliser sur la résolution algorithmique du problème du mot.

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 $\mathcal{B}_n$.

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. □

Corollaire 1.2.1. *Il est possible d'engendrer* $\mathfrak{S}_n$ *avec deux générateurs.*

Démonstration. Par théorème, $\mathcal{B}_n$ est engendré par deux éléments. De plus, il existe une morphisme surjectif déjà évoqué de $\mathcal{B}_n$ dans $\mathfrak{S}_n$. □

On constate ainsi que la théorie des tresses permet de démontrer des vérités dans d'autres champs. Au-delà, elle permet aussi d'établir des vérités dans la réalité. En effet, on peut par exemple montrer que $\mathcal{B}_n$ est sans torsion [8],

8. Un groupe est dit sans torsion si tous ses éléments différents du neutre sont d'ordre infini.

ce qui signifie, en particulier, que la reproduction d'un motif indénouable, par exemple des les cheveux de quelqu'un, ne se dénouera pas quelque soit le nombre de répétitions de ce motif.

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 [9], 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 la dénouabilité d'une tresse donnée. Pour continuer, on va utiliser les relations du groupe des tresses pour réarranger les mots en une forme plus simple à étudier.

Théorème 2.1.1. *Déterminer la dénouabilité d'une tresse revient à résoudre le problème du mot.*

Démonstration. Soit α et β deux tresses. Résoudre le problème du mot revient à savoir si α se déforme en β, c'est-à-dire si $\alpha\beta^{-1}$ se dénoue. En effet, $\alpha = \beta \iff \alpha\beta^{-1} = \epsilon$. □

Maintenant le problème réduit, on peut chercher à simplifier l'étude des tresses en triant leurs croisements. On introduit ainsi la réécriture qui a pour but de ranger les générateurs à gauche de la tresses et leurs inverses à droite. Cette réécriture est définie en deux temps, d'abord une étape élémentaire puis la succession de ces étapes.

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 tresses α et β, $\alpha \to^\star \beta$ si et seulement si il existe une suite finie de mots $(\gamma_k)_{k\in[\![0,n]\!]}$ telle que $\gamma_0 = \alpha$, $\gamma_n = \beta$ et, pour tout $k \in [\![0, n-1]\!]$, $\gamma_k \to \gamma_{k+1}$, c'est-à-dire si on peut passer de α à β par un nombre fini de réécritures directes successives.

9. Cet algorithme a pour défaut majeur qu'il n'est pas constructiviste, c'est-à-dire qu'il permet de déterminer si on peut déformer une tresse en une autre mais ne donne aucun moyen de le faire, contrairement aux autres algorithmes évoqués qui, eux, explicitent une déformation possible.

Remarque. On vérifie par une récurrence immédiate [10] que pour deux tresses α et β de $\mathcal{B}_n$ tels que $\alpha \to^\star \beta$, on a $\alpha = \beta$, de sorte qu'en réécrivant une tresse donnée, on soit toujours assurer de parler de la même.

Maintenant le système de réécriture établi, il s'agit d'étudier ses propriétés qui devraient aider à résoudre le problème du mot. Pour pouvoir l'utiliser, il faut avant tout s'assurer qu'il converge. On s'intéresse ensuite à la forme finale qu'on va obtenir.

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

Lemme 2.1.2 (Confluence)**.** *Si pour une tresse α de $\mathcal{B}_n$ donné, il existe deux tresses β et β' tels que $\alpha \to^\star \beta$ et $\alpha \to^\star \beta'$, alors il existe un mot γ de $\mathcal{B}_n$ tel que $\beta \to^\star \gamma$ et $\beta' \to^\star \gamma$.*

La confluence permet d'être certain que, même si plusieurs chemins de réécriture sont possibles, le résultat vers lequel cette dernière converge est unique.

Théorème 2.1.2 (Existence et unicité de la décomposition d'Ore)**.** *Pour toute tresse $\gamma \in \mathcal{B}_n$, il existe deux uniques tresses positives [11] α et β tels que $\gamma \to^\star \alpha\beta^{-1}$.*

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 le résultat de la réécriture qui garantirait la dénouabilité de la tresse.

Corollaire 2.1.1 (Théorème d'Ore)**.** *Pour deux tresses positives α et β, $\alpha\beta^{-1} = \epsilon$ si et seulement si $\alpha^{-1}\beta \to^\star \epsilon$.*

Démonstration. L'idée du théorème d'Ore est que, si on a trié les générateurs du côté gauche α et leurs inverses du côté droit β, alors en retournant les deux côtés de la tresse est en la réécrivant, tous les croisements vont se rencontrer de sorte que si, et seulement si la tresse est dénouable, tous les croisements disparaissent en rencontrant leur inverse. □

On peut donc résoudre le problème du mot en écrivant un algorithme qui réécrirait un mot donné en sa décomposition d'Ore. Un tel algorithme

10. Étant donné que la réécriture directe $\to$ se base sur des cas particuliers des relations du groupe des tresses qui conserve leur égalité.

11. Une tresse positive est une tresse qui ne s'écrit qu'avec des générateurs et aucun inverse.

de réécriture complété avec l'identification de la décomposition est appelé algorithme de retournement. Il est à noter que, pour se faire, la réécriture devra être utilisé deux fois. La première pour obtenir la décomposition, la seconde pour vérifier ou non la dénouabilité par théorème.

2.2 Implémentation.

On se propose d'implémenter l'algorithme de retournement en OCaml pour en faire une rapide 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 le passage à l'opposé de tous les entiers.

On est alors prêt à programmer l'algorithme de retournement. Pour se faire, on écrit une sous-fonction qui parcourt la liste, stocke chaque entier passé, applique une relation de réécriture directe dès que nécessaire puis recommence son parcours un entier en arrière pour finalement renvoyer l'ensemble des lettres parcourue et réécrite. Elles seront cependant dans le mauvais sens. Il faut donc les retourner avec un "miroir" pour obtenir le mot réécrit.

On complète avec une fonction séparant les deux tresses positives de la décomposition d'Ore d'une tresse réécrite, en parcourant chaque entier jusqu'à s'arrêter à la séparation entre les deux, c'est-à-dire le premier entier négatif. 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 exclusivement en des appels de ces quatre fonctions auxiliaires.

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 puisqu'il faut parcourir au moins une fois chaque entier de la liste pour la terminer. Pour autant, la majorité des appels consistent en l'application des différentes opérations de réécriture avec des retours en arrière dans la lecture de la liste, augmentant considérablement le nombre d'opérations à faire.

Au-delà de la longueur de la tresse, ses croisements et leur agencement sont centraux rendant difficile la détermination de la complexité moyenne mathématiquement. En conséquence, on se propose de procéder à des mesures expérimentales du temps d'exécution moyen d'une réécriture.

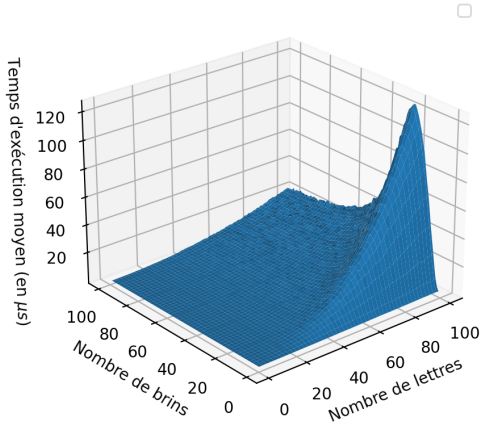

Figure 5 – Expérience réalisée avec un nombre de mots et de brins variant indépendamment de 1 à 100.

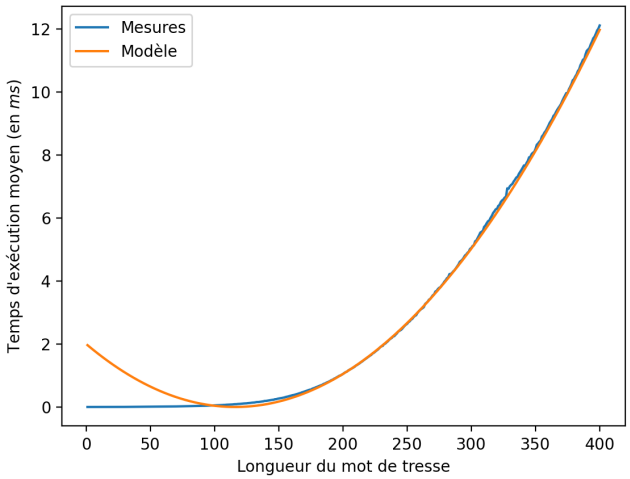

FIGURE 6 – Expérience réalisée avec un nombre de mots variant de 1 à 400 avec un nombre de brins fixé à 40 comparée à une parabole.

On constate que les mesures de complexité sont bien approchées par une parabole asymptotiquement. On peut conjecturer que la complexité est quadratique, ce que l'on peut démontrer[9], non sans mal. Pour l'améliorer, on pourra vérifier si la tresse donnée en entrée est pure, condition nécessaire pour être vide. Ce test est possible en complexité linéaire, permettant 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 réduit de la sorte considérablement la place en mémoire prise par un générateur. Il faut cependant vérifier que ce changement n'augmente pas trop la taille des listes, fait inévitable en réduisant le nombre de générateurs, sans quoi la nouvelle structure de données prendrait plus de place que l'originale proposée. On accepte de plus une augmentation du temps de calcul au profit de la mémoire, ce qui peut être discutable.

Listing 7 – Structure de données avec deux générateurs

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

Dans tous les cas, on peut conclure que, même si le problème du mot a été résolu, la théorie des tresses est loin d'avoir délivrée tous ses secrets. Il reste de nombreuses propriétés restent à découvrir et étudier, que ce soit pour affiner l'algorithme, en établir d'autres, constructivistes par exemple, ou encore établir des preuves mathématiques dans d'autres champs de recherche comme on a pu le faire avec les groupes symétriques.

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.