Calcul efficace du shuffle de k mots

Cyril Allauzen

Institut Gaspard-Monge, Université de Marne-la-Vallée,
Cité Descartes, Champs-sur-Marne,
F-77454 Marne-la-Vallée Cedex 2, France.

allauzen@univ-mlv.fr

Résumé

Nous présentons dans la suite un algorithme de calcul du shuffle (ou mélange) de k mots u_1, u_2,..., u_k dont la complexité est en $O\left(\binom{|u_1|+\cdots+|u_k|}{|u_1|,\ldots,|u_k|}\right)$ en temps et en espace. Cet algorithme construit une représentation du shuffle linéaire en fonction du cardinal de celui-ci.

1 Introduction

Un *mot* u est une suite finie de lettres prises dans un alphabet fini Σ. On note Σ^* l'ensemble des mots formés de lettres de Σ. La longueur du mot u est notée $|u|$. On note $u[i]$ la i-ème lettre du mot u pour $1 \leq i \leq |u|$ et, pour $\sigma \in \Sigma$, $|u|_\sigma$ le nombre d'occurrences de la lettre σ dans u.

Le *shuffle* ou *mélange* de deux mots u et v est l'ensemble, noté $u \sqcup\!\sqcup\, v$, défini par

$$\begin{aligned} u \sqcup\!\sqcup\, v \quad = \quad \{\, h \in \Sigma^* \mid \quad & h = u_1v_1u_2v_2\cdots u_nv_n \quad n \geq 0 \quad u_i, v_i \in \Sigma^* \\ & u = u_1u_2\cdots u_n \quad v = v_1v_2\cdots v_n \}. \end{aligned}$$

On peut remarquer que le nombre de mots dans le shuffle $u \sqcup\!\sqcup\, v$ est borné par $\binom{|u|+|v|}{|u|}$.

La définition du shuffle se généralise facilement à un ensemble de mots. Le shuffle des mots u_1, u_2,..., u_k, noté $\sqcup\!\sqcup_1^k\, u_i$, est l'ensemble des mots h tel qu'il existe un entier $n \geq 0$ et des mots $u_j^i \in \Sigma^*$ pour $1 \leq i \leq k$ et $1 \leq j \leq n$ vérifiant $u_i = u_1^i u_2^i \cdots u_n^i$ pour tout i et $h = u_1^1 \cdots u_1^k u_2^1 \cdots u_2^k \cdots\cdots u_n^1 \cdots u_n^k$. Le nombre de mots dans le shuffle $\sqcup\!\sqcup_1^k\, u_i$ est borné par le coefficient multinomial $\binom{|u_1|+\cdots+|u_k|}{|u_1|,\ldots,|u_k|}$.

Il existait jusqu'à présent peu d'algorithmes efficaces de calcul du shuffle. Le plus efficace pour deux mots, l'algorithme de Spehner [3], a une complexité en $O((|u|+|v|)\binom{|u|+|v|}{|u|})$. Nous présentons dans la suite un algorithme de calcul du shuffle de k mots en $O\left(\binom{|u_1|+\cdots+|u_k|}{|u_1|,\ldots,|u_k|}\right)$ et donc de deux mots en $O\left(\binom{|u|+|v|}{|u|}\right)$. Cet algorithme construit une représentation du shuffle linéaire en fonction du cardinal de celui-ci.

2 Shuffle et partitions

On note $E_n = \{1, 2, \ldots, n\}$ l'ensemble des n premiers entiers positifs. Une k-partition de taille $(p_1, p_2, \ldots, p_k)$ de E_n est une partition de E_n en k sous-ensembles de cardinaux respectifs p_1,p_2,…,p_k. Le nombre de k-partitions de E_n de taille $(p_1, p_2, \ldots, p_k)$ est donné par le coefficient multinomial $\binom{n}{p_1, p_2, \ldots, p_k}$.

On peut représenter une telle k-partition par un mot de E_k^* de longueur n. Le mot $x \in E_k^*$ représente la partition $P = (P_i)_{i=1\ldots k}$ avec $P_i = \{j | x[j] = i\}$. L'ensemble des k-partitions de E_n de taille $(p_1, p_2, \ldots, p_k)$ est alors représenté par l'ensemble des mots de E_k^* où chaque lettre i apparait p_i fois.

Lemme 1 *Soit u_1,u_2,…,u_k k mots de longueurs respectives p_1,p_2,…,p_k. Alors, il existe une surjection de l'ensemble des k-partitions de taille $(p_1, p_2, \ldots, p_k)$ de $E_{p_1+\cdots+p_k}$ sur le shuffle des u_i.*

Preuve. Considérons k mots u_1,u_2,…,u_k de longueurs respectives p_1,p_2,…,p_k. Posons $n = p_1 + p_2 + \cdots + p_k$. Posons $z_j^i = |x[1 \ldots j]|_i$.

Soit f la fonction de l'ensemble des k-partitions de E_n de taille $(p_1, p_2, \ldots, p_k)$ dans le shuffle $\sqcup\!\sqcup_1^k u_i$ définie de la façon suivante. A chaque mot x repésentant une k-partition de E_n de taille $(p_1, p_2, \ldots, p_k)$, on associe le mot $h \in \Sigma^*$ défini par $h[i] = u_{x[i]}[z_i^{x[i]}]$ pour $1 \leq i \leq n$.

Montrons que h appartient bien à $\sqcup\!\sqcup_1^k u_i$. Considérons les mots $u_j^i \in \Sigma^*$ défini par :

$$u_j^i = \begin{cases} h[j] & \text{si } x[j] = i, \\ \epsilon & \text{sinon.} \end{cases}$$

On obtient bien alors que $h = u_1^1 \cdots u_1^k u_2^1 \cdots u_2^k \cdots\cdots u_n^1 \cdots u_n^k$ et que, pour tout i, $u_i = u_1^i u_2^i \cdots u_n^i$. Ce qui prouve que $h \in \sqcup\!\sqcup_1^k u_i$.

La fonction f est donc bien définie. Montrons maintenant qu'elle est surjective. Soit h un mot appartenant au shuffle $\sqcup\!\sqcup_1^k u_i$. Par définition, h s'écrit $h = u_1^1 \cdots u_1^k u_2^1 \cdots u_2^k \cdots\cdots u_n^1 \cdots u_n^k$ avec, pour tout i, $u_i = u_1^i u_2^i \cdots u_n^i$. Considérons maintenant le mot de E_k^* défini par :

$$x = 1^{|u_1^1|} 2^{|u_1^2|} \ldots k^{|u_1^k|} 1^{|u_2^1|} 2^{|u_2^2|} \ldots k^{|u_2^k|} \ldots\ldots 1^{|u_n^1|} 2^{|u_n^2|} \ldots k^{|u_n^k|}.$$

Alors comme, pour tout i, $|u_i| = |u_1^i| + |u_2^i| + \cdots + |u_n^i| = p_i$, on a $|x|_i = p_i$ et x est bien le codage d'une k-partition de taille $(p_1, \ldots, p_k)$. De plus, il apparait clairement que $f(x) = h$, ce qui prouve la surjectivité de f. □

L'existence de cette surjection nous permet d'établir la majoration suivante :

$$\left| \mathop{\sqcup\!\sqcup}_{i=1}^{k} u_i \right| \leq \binom{p_1 + p_2 + \cdots + p_k}{p_1, p_2, \ldots, p_n}$$

3 Représentations

3.1 Représentation des k-partitions d'un ensemble

Si l'on représente chaque k-partition par un mot de E_k^*, la liste des k-partitions de taille $(p_1, \ldots, p_k)$ nécessite $(p_1 + \cdots + p_k) \times \binom{p_1+\cdots+p_k}{p_1,\ldots,p_k}$ caractères.

Nous allons voir un moyen de représenter l'ensemble des k-partitions de taille $(p_1, \ldots, p_k)$ en $O\left(\binom{p_1+\cdots+p_k}{p_1,\ldots,p_k}\right)$.

Pour cela, nous commençons par construire l'arbre lexicographique (on dit aussi arbre digital ou trie) de l'ensemble des mots de E_k^* qui représentent une k-partition de taille $(p_1, \ldots, p_k)$. On note $\mathcal{T}(p_1, \ldots, p_k)$ cet arbre.

Lemme 2 *Le degré de la racine de $\mathcal{T}(p_1, \ldots, p_k)$ est k. Le degré d'un nœud possédant un père de degré r est r ou $r - 1$.*

Preuve. Pour chaque nœud t de $\mathcal{T}(p_1, \ldots, p_k)$, on note $d(t)$ son degré, $x(t)$ le mot de E_k^* étiquetant le chemin qui mène de la racine au nœud t et $X(t) = \{i \in E_k \mid |x(t)_i| \neq p_i\}$. Si t a pour fils f par i, alors $x(f) = x(t) \cdot i$, $|x(f)|_j = |x(t)_j|$ si $j \neq i$ et $|x(f)|_i = |x(t)_i| + 1$. On en déduit que $X(t) \backslash \{i\} \subseteq X(f) \subseteq X(t)$.

Comme l'ensemble $X(t)$ correspond aux lettres que l'on peut et que l'on doit utiliser dans la suite de l'arbre, il apparait clairement que (1) t doit avoir une arête sortante pour chaque lettre de $X(t)$ et que (2) si t possède une arête sortante pour une lettre i alors $i \in X(t)$. On en déduit que $d(t) = |X(t)|$. Et ainsi, si f est un fils de t, alors $d(t) - 1 \leq d(f) \leq d(t)$. De plus, si t est la racine, $X(t) = E_k$ et $d(t) = k$. $\Box$

On appelle *chemin de degré 1* un chemin dont tous les nœuds intermédiaires (extrémités excluses) sont de degré 1. Un *chemin maximal de degré 1* est un chemin de degré 1 de longueur maximale.

Lemme 3 *Les chemins maximaux de degré 1 dans $\mathcal{T}(p_1, \ldots, p_k)$ mènent aux feuilles de l'arbre. De plus, chacun de ces chemins possède une étiquette de la forme ij^l avec $i, j \in E_k$ et $l \geq 0$.*

Preuve. Reprenons les notations de la preuve précédente : $d(t)$, $x(t)$ et $X(t)$ pour chaque nœud t de $\mathcal{T}(p_1, \ldots, p_k)$.

Soit $c = (c_0, c_1, \ldots, c_{l+1})$ un chemin maximal de degré 1 où les c_i sont des nœuds de l'arbre et $l \geq 0$. Comme c est un chemin maximal de degré 1, on a $d(c_0) > 0$, $d(c_\alpha) = 1$ si $1 \leq \alpha \leq l$ et $d(c_l) = 0$. Comme le degré diminue d'au plus 1 quand on passe d'un nœud à un de ses fils (lemme 2), on a nécessairement $d(c_0) = 2$ et donc $|X(c_0)| = 2$. On peut donc poser $X(c_0) = \{i, j\}$. Comme $|X(c_1)| = 1$, $X(c_1)$ ne possède plus qu'un élément : j par exemple. On en déduit que $|x(c_0)|_i = p_i - 1$ et que l'arête de c_0 à c_1 est étiquetée par i. Alors, comme $X(c_1) = \{j\}$, l'arête de c_1 à c_2 est étiquetée par j. De même pour tout $2 \leq \alpha \leq l$, $X(c_\alpha) = \{j\}$ et l'arête de c_α à $c_{\alpha+1}$ est étiquetée par j. Le chemin c est donc étiqueté par le mot ij^l. $\Box$

L'opération de *compactage* d'un arbre consiste à remplacer chaque chemin maximal de degré 1 par une seule arête étiquetée par le mot qui étiquette ce chemin. Cette opération est par exemple couramment utilisée sur les arbres des suffixes.

Nous appelons $T(p_1, \ldots, p_k)$ l'arbre obtenu en compactant l'arbre $\mathcal{T}(p_1, \ldots, p_k)$.

Lemme 4 *L'arbre $T(p_1, \ldots, p_k)$ peut être codé en $O\left(\binom{p_1+\cdots+p_k}{p_1,\ldots,p_k}\right)$.*

Preuve. D'après le lemme 3, les seuls chemins de $\mathcal{T}(p_1, \ldots, p_k)$ et donc les arêtes compactées de $T(p_1, \ldots, p_k)$ possèdent des étiquettes de la forme ij^l avec $i, j \in E_k$ et $l \geq 0$. Il est donc possible de représenter ces étiquettes en espace constant (deux lettres plus un entier).

On peut maintenant remarquer que, dans l'arbre $T(p_1, \ldots, p_k)$, tous les nœuds internes ont un degré supérieur ou égal à 2. Or dans un tel arbre, le nombre total de nœuds est borné par deux fois le nombre de feuilles. Etant donné que l'arbre $T(p_1, \ldots, p_k)$, tout comme l'arbre $\mathcal{T}(p_1, \ldots, p_k)$, posséde $\binom{p_1+\cdots+p_k}{p_1,\ldots,p_k}$ feuilles, le nombre total de ses nœuds est borné par $2 \cdot \binom{p_1+\cdots+p_k}{p_1,\ldots,p_k}$. De plus, chaque nœud et chaque arête pouvant être codés en espace constant, l'arbre $T(p_1, \ldots, p_k)$ peut être codé en $O\left(\binom{p_1+\cdots+p_k}{p_1,\ldots,p_k}\right)$. □

3.2 Génération des k-partitions

Nous présentons maintenant un algorithme permettant de constuire l'arbre $T(p_1, \ldots, p_k)$ en temps $O\left(\binom{p_1+\cdots+p_k}{p_1,\ldots,p_k}\right)$.

Lemme 5 *L'algorithme dont le pseudo-code est donné figure 1 construit, sur la donnée d'un k-uplet $(p_1, p_2, \ldots, p_k)$, l'arbre $T(p_1, \ldots, p_k)$ représentant les k-partitions de taille $(p_1, p_2, \ldots, p_k)$. La complexité de cette algorithme est en $O\left(\binom{p_1+\cdots+p_k}{p_1,\ldots,p_k}\right)$ en temps et en espace.*

Preuve. Lorsque la fonction **Construire_Rec_Arbre** est appelée sur un nœud r, l'ensemble X défini ligne 1 correspond bien à l'ensemble $X(r)$ de la preuve du lemme 2. Ainsi les transitions qui sont créées par cette fonction ligne 6, 9, 11 sont bien les bonnes. De plus, l'algorithme détecte, grâce à l'ensemble Y, les chemins de degré 1 et les génère directement sous forme compactée.

Pour chaque nœud de l'arbre, l'algorithme effectue un nombre constant d'opérations. La complexité en temps est donc $O\left(\binom{p_1+\cdots+p_k}{p_1,\ldots,p_k}\right)$. L'espace mémoire nécessaire est borné par la taille de l'arbre construit. □

3.3 Représentation du shuffle

Considérons k mots u_1,u_2,...,u_k sur Σ de longueurs respectives p_1,p_2,...,p_k. Nous allons représenter l'ensemble $\sqcup\!\sqcup_1^k u_i$ sous forme d'un arbre lexicographique que nous notons $\mathcal{A}(u_1, u_2, \ldots, u_k)$.

La surjection f de l'ensemble des k-partitions de taille $(p_1, p_2, \ldots, p_k)$ sur le shuffle $\sqcup\!\sqcup_1^k u_i$ peut être utiliser pour associer, à chaque chemin partant de la racine de $\mathcal{T}(p_1, \ldots, p_k)$

```
Construire_Arbre(p_1, p_2, ..., p_k)
    créer arbre T
    r ← racine de T
    Construire_Rec_Arbre(r, p_1, p_2, ..., p_k)

Construire_Rec_Arbre(r, x_1, x_2, ..., x_k)
    X ← {i | x_i ≠ 0}
    Y ← {i | x_i = 1}
    si |X| > 2 ou (|X| = 2 et |Y| = 0)
        pour tout i ∈ X faire
            créer un fils f de r par i
            Construire_Rec_Arbre(f, x_1, ..., x_i − 1, ..., x_k)
    sinon /* |X| = 2 et |Y| = 1*/
            on note Y = {i} et X = {i, j}
            créer un fils f de r par j
            Construire_Rec_Arbre(f, x_1, ..., x_j − 1, ..., x_k)
            créer une feuille f', fille de r par ij^{x_j}
```

FIG. 1 – Pseudo-code de l'algorithme de construction de l'arbre $T(p_1, \ldots, p_k)$ représentant les k-partitions de taille $(p_1, p_2, \ldots, p_k)$.

et arrivant à une de ses feuilles, un chemin partant de la racine de $\mathcal{A}(u_1, u_2, \ldots, u_k)$ et arrivant à une de ses feuilles. La surjection f induit ainsi une surjection f' de l'ensemble des nœuds de $\mathcal{T}(p_1, \ldots, p_k)$ sur celui des nœuds de $\mathcal{A}(u_1, u_2, \ldots, u_k)$: au i-ème nœud sur un chemin c partant de la racine de $\mathcal{T}(p_1, \ldots, p_k)$ et arrivant à une de ses feuilles, on associe le i-ème nœud sur le chemin $f(c)$.

Nous allons maintenant appliquer l'opération de compactage sur l'arbre $\mathcal{A}(u_1, u_2, \ldots, u_k)$ mais cette fois-ci nous ne l'appliquerons qu'à certains chemins : les seuls chemins compactés seront les chemins de degré 1 de $\mathcal{A}(u_1, u_2, \ldots, u_k)$ qui sont les images par f' des chemins maximaux de degré 1 dans $\mathcal{T}(p_1, \ldots, p_k)$. L'arbre ainsi obtenu est noté $A(u_1, u_2, \ldots, u_k)$.

Lemme 6 *Les étiquettes des arêtes compactées de $A(u_1, u_2, \ldots, u_k)$ peuvent être codée en espace constant.*

Preuve. Les arêtes compactées de $A(u_1, u_2, \ldots, u_k)$ sont les images par la surjection f' d'arêtes compactées de $T(p_1, \ldots, p_k)$. Ces dernières étant étiquetées ij^l avec $l > 0$ et $i \neq j \in E_k$ et menant à des feuilles, les arêtes compactées de $A(u_1, u_2, \ldots, u_k)$ sont étiquetées par $u_i[p_i]u_j[p_j - l] \cdots u_j[p_j]$, c'est-à-dire par la dernière lettre d'un u_i suivie par un suffixe d'un u_j distincts de u_i. Elles peuvent donc être codées par les trois indices (i, j, l). $\square$

Théorème 1 *L'arbre $A(u_1, u_2, \ldots, u_k)$ peut être codé en $O\left(\binom{p_1+\cdots+p_k}{p_1,\ldots,p_k}\right)$.*

Preuve. L'existance de la surjection f' implique que $A(u_1, u_2, \ldots, u_k)$ possède moins de nœuds que $T(p_1, \ldots, p_k)$. Le nombre de nœuds de $A(u_1, u_2, \ldots, u_k)$ est donc borné par

$\binom{p_1+\cdots+p_k}{p_1,\ldots,p_k}$. Chaque nœud et chaque arête pouvant être codés en espace constant, on obtient un codage de $A(u_1, u_2, \ldots, u_k)$ en $O\left(\binom{p_1+\cdots+p_k}{p_1,\ldots,p_k}\right)$. $\square$

Si l'on décide maintenant de compacter l'ensemble des chemins maximaux de degré 1 de $\mathcal{A}(u_1, u_2, \ldots, u_k)$, on obtient un nouvel arbre que l'on note $A'(u_1, u_2, \ldots, u_k)$.

Lemme 7 *Considérons une arête compactées de $A'(u_1, u_2, \ldots, u_k)$.*

(i) Son étiquette peut s'écrire $\sigma w_{i_1} \ldots w_{i_l}$ avec $\sigma \in \Sigma$, $l > 0$ où les i_j sont tous distincts et w_{i_j} est un facteur non vide de u_{i_j}.

(ii) Dans le cas particulier où $l > 1$, cette étiquette s'écrit plus simplement $\sigma\tau^m$ avec $m > 0$ et $\sigma, \tau \in \Sigma$.

Preuve. Considérons une chemin γ maximal de degré 1 dans l'arbre $\mathcal{A}(u_1, u_2, \ldots, u_k)$. Notons s le premier nœud de γ et t le dernier. Le chemin γ se trouve sur au moins sur un chemin menant de la racine de $\mathcal{A}(u_1, u_2, \ldots, u_k)$ à une de ses feuilles. Considérons un mot $h \in \sqcup\!\sqcup_1^k u_i$ correspondant à un tel chemin Γ. Soit e l'étiquette du chemin γ. Le mot h peut alors s'écrire : $h = h_1 e h_2$. Considérons maintenant un mot $x \in E_k^*$ dont l'image par f est h : $h = f(x)$. Ce mot x peut s'écrire $x = x_1 y x_2$ avec $|x_1| = |h_1|$, $|x_2| = |h_2|$ et $|y| = |e|$.

Le mot x permet d'attribuer à chaque lettre de h la lettre du u_i dont elle provient. Ainsi, à chaque lettre de e ont peut associé la lettre du u_i dont elle provient. Posons $e = \sigma e'$ avec $\sigma \in \Sigma$ et $y = y_0 y'$ avec $y_0 \in E_k$.

Pour chaque $i \in E_k$, considérons le mot w_i formé des lettres de e' attribuées au mot u_i par le codage x (prise dans l'ordre de leur apparition dans e'). Le mot w_i est par définition un facteur de u_i. On a $w_i = \epsilon$ si aucune lettre de e' n'est attribuée à u_i *i.e.* si $|y'|_i = 0$. On a aussi $|w_i| = |y'|_i$ pour tout $i \in E_k$.

Considérons maintenant le mot $x' = x_1 y_0 1^{|w_1|} 2^{|w_2|} \cdots k^{|w_k|} x_2$. Comme $|w_i| = |y'|_i$ pour tout i, le mot x' représente bien une k-partition de E_k, et ainsi le mot $h' = f(x') = h_1 \sigma w_1 w_2 \cdots w_k h_2$ appartient à $\sqcup\!\sqcup_1^k u_i$.

Regardons à quel chemin dans $\mathcal{A}(u_1, u_2, \ldots, u_k)$ correspond le mot h'. La lecture du prefixe h_1 de h' mène jusqu'en s. Or en s, il n'y a qu'une seule transition étiquetée par σ. On est donc obliger d'emprunter le chemin γ jusqu'en t. En t, il reste à lire h_2. Ainsi, le mot h' correpond lui aussi au chemin Γ de l'arbre $\mathcal{A}(u_1, u_2, \ldots, u_k)$ et $\sigma w_1 w_2 \cdots w_k$ est donc aussi l'étiquette du chemin γ. On a donc obtenu que $e = \sigma w_1 w_2 \cdots w_k$ soit $e' = w_1 w_2 \cdots w_k$. Posons $\{i \in E_k || w_i| \neq 0\} = \{i_1, i_2, \ldots, i_l\}$. On peut alors écrire $e = \sigma w_{i_1} \ldots w_{i_l}$. Ce qui prouve le point (i).

Supposons maintenant que $l \geq 2$. Posons, pour $1 \leq \alpha, \beta \leq l$,

$$e_\alpha = w_{i_\alpha} w_{i_1} \ldots w_{i_{\alpha-1}} w_{i_{\alpha+1}} \ldots w_{i_l} \text{ et}$$

$$e_{\alpha,\beta} = w_{i_\beta}[1] w_{i_\alpha} w_{i_1} \ldots w_{i_{\alpha-1}} w_{i_{\alpha+1}} \ldots w_{i_{\beta-1}} w_{i_\beta}[2 \ldots |w_{i_\beta}|] w_{i_{\beta+1}} \ldots w_{i_l}.$$

Pour tout α, le mot $h_1 \sigma e_\alpha h_2$ appartient au shuffle des u_i et si $\bar{\alpha} \neq \alpha$, le mot $h_1 \sigma e_{\alpha,\bar{\alpha}} h_2$ appartient aussi au shuffle. Pour la même raison que précédement, ces mots correspondent tous au même chemin Γ. On en déduit que pour tout α, β, $e_\alpha = e_\beta$ et que pour tout α, si $\bar{\alpha} \neq \alpha$, $e_{\alpha,\bar{\alpha}} = e_\alpha$.

Or $e_\alpha = e_\beta$ implique $e_\alpha[j] = e_\beta[j]$ et donc

$$w_{i_\alpha}[j] = w_{i_\beta}[j] \tag{1}$$

pour tout $j \leq \min(|w_{i_\alpha}|, |w_{i_\beta}|)$. Et de $e_{\alpha,\bar{\alpha}} = e_\alpha$ si $\bar{\alpha} \neq \alpha$, on déduit que $e_{\alpha,\bar{\alpha}}[j+1] = e_\alpha[j+1]$ et

$$w_{i_\alpha}[j] = w_{i_\alpha}[j+1] \tag{2}$$

pour tout $j < |w_{i_\alpha}|$.

De (2), on déduit qu'il existe $\tau_\alpha \in \Sigma$ tel que $w_{i_\alpha}[j] = \tau_\alpha$ pour tout $j \leq |w_{i_\alpha}|$. Enfin de (1), on déduit que $\tau_\alpha = \tau_\beta = \tau$ quelque soit α et β.

On obtient donc que $e' = \tau^m$ soit $e = \sigma\tau^m$ avec $m = |e'| > 0$. Ce qui prouve le point (ii). □

Corollaire 1 *Les étiquettes des arêtes compactées de $A'(u_1, u_2, \ldots, u_k)$ peuvent être codées en espace constant.*

Preuve. Considérons une arête compactée de $A'(u_1, u_2, \ldots, u_k)$ et son étiquette. Considérons la décomposition de cette étiquette donnée lemme 7.

Si $l = 1$, alors cette étiquette consiste en une lettre suivie d'un facteur d'un u_i. Cette étiquette est donc codable par sa première lettre suivi de trois indices (quel u_i, indice de début et de fin du facteur dans u_i).

Si $l > 1$, alors cette étiquette s'écrit $\sigma\tau^m$ et est donc codable par les deux lettres σ et τ et l'entier m.

Dans les deux cas, l'étiquette peut être codée en espace constant. □

Théorème 2 *L'arbre $A'(u_1, u_2, \ldots, u_k)$ peut être codé en $O\left(|\sqcup\!\sqcup_1^k u_i|\right)$.*

Preuve. On peut maintenant remarquer que, dans l'arbre $A'(u_1, u_2, \ldots, u_k)$, tous les nœuds internes ont un degré supérieur ou égal à 2. Or dans un tel arbre, le nombre total de nœuds est borné par deux fois le nombre de feuilles. Etant donné que l'arbre $A'(u_1, u_2, \ldots, u_k)$ possède une feuille pour chaque mot appartenant au shuffle de u_i soit $|\sqcup\!\sqcup_1^k u_i|$ feuilles, le nombre total de ces nœuds est borné par $2 \cdot |\sqcup\!\sqcup_1^k u_i|$. De plus, chaque nœud et chaque arête pouvant être codés en espace constant, l'arbre $A'(u_1, u_2, \ldots, u_k)$ peut être codé en $O\left(|\sqcup\!\sqcup_1^k u_i|\right)$. □

4 Algorithme de calcul du shuffle

Théorème 3 *L'algorithme dont le pseudo-code est donné figure 2 construit, sur la donnée de k mots $u_1, u_2, \ldots, u_k$, l'arbre $A(u_1, \ldots, u_k)$.*

La complexité de cette algorithme est $O\left(\binom{p_1+\cdots+p_k}{p_1,\ldots,p_k}\right)$ en temps et en espace.

Preuve. La validité de l'algorithme découle de la validité de l'algorithme de construction de l'arbre $T(p_1, \ldots, p_n)$ (lemme 5).

En effet, cet algorithme suit le même schema que le précédent. L'idée principale est de générer les k-partitions mais cette fois-ci en étiquetant les chemins par les mots du shuffle.

Construire_Shuffle$(u_1, u_2, \ldots, u_k)$
    créer arbre T
    $r \leftarrow$ racine de T
    pour $i = 1$ à k faire $p_i = |u_i|$
    Construire_Rec_Shuffle$(r, u_1, \ldots, u_k, p_1, \ldots, p_k)$

Construire_Rec_Shuffle$(r, u_1, \ldots, u_k, x_1, \ldots, x_k)$
    $X \leftarrow \{i | x_i \neq 0\}$
    $Y \leftarrow \{i | x_i = 1\}$
    si $|X| > 2$ ou $(|X| = 2$ et $|Y| = 0)$
        pour tout $i \in X$ faire
            $f \leftarrow$ Chercher_Fils$(r, i, u_1, \ldots, u_k, x_1, \ldots, x_k)$
            si f défini
                Construire_Rec_Shuffle$(f, u_1, \ldots, u_k, x_1, \ldots, x_i - 1, \ldots, x_k)$
    sinon /* $|X| = 2$ et $|Y| = 1$*/
        on note $Y = \{i\}$ et $X = \{i, j\}$
            $f \leftarrow$ Chercher_Fils$(r, j, u_1, \ldots, u_k, x_1, \ldots, x_k)$
            si f défini
                Construire_Rec_Shuffle$(f, u_1, \ldots, u_k, x_1, \ldots, x_j - 1, \ldots, x_k)$
        si r n'a pas de fils par une arête dont l'étiquette est ou commence par $u_i[p_i - x_i + 1]$
            créer une feuille f' fille de r par $u_i[p_i]u_j[p_j - x_j] \cdots u_j[p_j]$
                étiquette codée par (i, j, x_j)

Chercher_Fils$(r, i, u_1, \ldots, u_k, x_1, \ldots, x_k)$
    si r a un fils par une arête étiquetée $u_i[p_i - x_i + 1]$
        $f \leftarrow$ ce fils
    sinon si r a un fils par une arête compactée codée (i', j', l')
            dont l'étiquette commence par $u_i[p_i - x_i + 1]$
        si Doit_Décompacter$(r, i, u_1, \ldots, u_k, x_1, \ldots, x_k, j', l')$
            décompacter cette arête
            $f \leftarrow$ le nouveau fils de r par l'arête simple étiquetée par $u_i[p_i - x_i + 1]$
        sinon $f \leftarrow$ `non_défini`
    sinon créer un fils f de r par $u_i[p_i - x_i + 1]$
    retourner f

Doit_Décompacter$(r, u_1, \ldots, u_k, x_1, \ldots, x_k, j', l', t)$
    $b \leftarrow$ `faux`
    si $t > l$ alors retourner `faux`
    $X \leftarrow \{j | x_j \neq 0\}$
    pour tout $j \in X$ faire
        si $u_j[p_j - x_j + 1] = u_{j'}[p_{j'} - l + t]$
            $b \leftarrow b$ ou Doit_Décompacter$(r, u_1, \ldots, u_k, x_1, \ldots, x_j - 1, \ldots, x_k, j', l', t + 1)$
        sinon retourner `vrai`
    retourner b

FIG. 2 – Pseudo-code de l'algorithme de construction de l'arbre $A(u_1, \ldots, u_k)$.

Les paramètres $(x_1, \ldots, x_n)$ de la fonction **Construire_Rec_Shuffle** permettent de simuler l'algorithme de construction de l'arbre $T(p_1, \ldots, p_n)$. Ils représentent le nombre de lettres de chacun des u_i qu'il reste à utiliser. Lorsque l'on atteint un nœud t, ces paramètres permettent de déterminer l'ensemble $X(t)$ correspondant au prefixe $x(t)$ de la k-partition que l'on est en train de générer (voir preuve du lemme 2). Pour chaque indice $i \in X(t)$, il faut que t est une transition par la lettre de u_i qui correspond : $u_i[p_i - x_i + 1]$. Mais cette fois-ci, deux indices distincts dans $X(t)$ peuvent correspondre à la même lettre de Σ. Et dans ce cas, il ne faut créer qu'une seule transition : c'est le rôle de la fonction **Chercher_Fils** de gérer celà. Enfin, tous les chemins de degré 1 dans l'arbre $T(p_1, \ldots, p_n)$ ne correspondent pas nécessairement à des chemins de degré 1 dans l'arbre $A(u_1, \ldots, u_k)$. C'est le rôle de la fonction **Doit_Décompacter** de les détecter.

A chaque passage sur un nœud donné de l'arbre $A(u_1, \ldots, u_k)$, l'algorithme effectue un nombre constant d'opérations. Mais, l'algorithme peut passer plusieures fois sur le même nœud. Cependant, le nombre de fois que l'on passe par un nœud t est égal au nombre de nœuds de l'arbre $T(p_1, \ldots, p_n)$ qui ont pour image par f' le nœud t. Le nombre d'opérations effectuées est donc proportionnel à la taille de l'arbre $T(p_1, \ldots, p_n)$. La complexité en temps est donc $O\left(\binom{p_1+\cdots+p_k}{p_1,\ldots,p_k}\right)$.

L'espace mémoire nécessaire est borné par la taille de l'arbre construit. □

Cet algorithme permet déjà de calculer efficacement le shuffle de k-mots. Cependant la représentation du shuffle qu'il construit - l'arbre $A(u_1, \ldots, u_k)$ - a une taille en $O\left(\binom{p_1+\cdots+p_k}{p_1,\ldots,p_k}\right)$. Il est possible de faire mieux : l'arbre $A'(u_1, \ldots, u_k)$ dont la taille est en $O\left(|\sqcup_1^k u_i|\right)$ en payant un coût supplémentaire en temps restant en $O\left(\binom{p_1+\cdots+p_k}{p_1,\ldots,p_k}\right)$.

Théorème 4 *L'algorithme dont le pseudo-code est donné figure 3 construit, sur la donnée de k mots $u_1, u_2, \ldots, u_k$, l'arbre $A'(u_1, \ldots, u_k)$ avec une complexité en $O\left(\binom{p_1+\cdots+p_k}{p_1,\ldots,p_k}\right)$ en temps et en espace.*

Preuve. Nous construisons d'abord l'arbre $A(u_1, \ldots, u_k)$ en utilisant l'algorithme précédent. Pour obtenir l'arbre $A'(u_1, \ldots, u_k)$, il suffit alors de faire un parcours en profondeur de $A(u_1, \ldots, u_k)$ en compactant les chemins de degré 1 résiduels. Cependant, pour pouvoir coder les étiquettes compactées conformément au lemme 7, il faut être en mesure de déterminer pour chaque lettre le mot u_i dont elle provient.

Pour cela, nous attribuons à chaque nœud t un prefixe d'une k-partitions correspondant au préfixe du mot du shuffle menant en t (grâce aux paramètres $(x_1, \ldots, x_k)$ et $(y_1, \ldots, y_k)$). En cas d'ambiguité, on attribue à une lettre le plus petit indice possible.

Pour ce qui est de la complexité, cet algorithme entraine, par rapport à l'algorithme précédent, un coût supplémentaire en temps en $O\left(\binom{p_1+\cdots+p_k}{p_1,\ldots,p_k}\right)$ dû au parcours en profondeur de l'arbre $A(u_1, \ldots, u_k)$. □

```
Construire_Shuffle_Compact(u_1, u_2, ..., u_k)
        créer arbre T
        r ← racine de T
        pour i = 1 à k faire p_i = |u_i|
        Construire_Rec_Shuffle(r, u_1, ..., u_k, p_1, ..., p_k)
        Compact_Arbre(r, u_1, ..., u_k, p_1, ..., p_k)

Compact_Arbre(r, u_1, ..., u_k, x_1, ..., x_k)
        X ← {i | x_i ≠ 0}
        pour toute lettre σ tel qu'il existe une arête par σ
                i ← le plus petit indice ∈ X tel que σ = u_i[p_i − x_i + 1]
                f ← le fils de r par σ
                si le degré de f est > 1
                        Compact_Arbre(f, u_1, ..., u_k, x_1, ..., x_i − 1, ..., x_k)
        sinon /* le degré de f est 1 */
            Compact_Chemin(r, i, f, (u_1, ..., u_k), (x_1, ..., x_i − 1, ..., x_k), (x_1, ..., x_i − 1, ..., x_k), 0, 0)

Compact_Chemin(r, i, f, (u_1, ..., u_k), (x_1, ..., x_i, ..., x_k), (y_1, ..., y_i, ..., y_k), cas, j)
        σ ← u_i[p_i − x_i]
        si le degré de f est > 1
            si cas = 0 alors ne rien faire
            sinon
                compacter le chemin de r à f
                si cas = 1 alors utiliser le codage σ, j, p_j − x_j + 1, p_j
                si cas = 2 alors utiliser le codage σ, u_j[p_j − x_j + 1], y_1 − x_1 + ··· + y_k − x_k
            Compact_Arbre(f, u_1, ..., u_k, y_1, ..., y_i, ..., y_k)
        sinon /* le degré de f est 1 */
            f ← le fils unique de r
            si l'étiquette de r → f est une lettre τ ∈ Σ
                α ← le plus petit indice de X tel que τ = u_α[p_α − y_α + 1]
                si cas = 0 alors cas ← 1 et j ← α
                si cas = 1 et j ≠ α alors cas ← 2
                Compact_Chemin(r, i, f, (u_1, ..., u_k), (x_1, ..., x_i, ..., x_k), (y_1, ..., y_α − 1, ..., y_k), cas, j)
            sinon /* l'arête r → f est compactée codée (i', j', l') */
                compacter le chemin de r à f et le coder σ, u_{j'}[p_{j'}], p_1 − x_1 + ··· + p_k − x_k
```

FIG. 3 – Pseudo-code de l'algorithme de construction de l'arbre $A'(u_1, \ldots, u_k)$.

5 Conclusion

Nous avons présenté le premier algorithme réellement efficace de calcul du shuffle de k mots. Sa complexité en temps et en espace est proportionnelle au coefficient multinomial associé aux longueurs des mots :

$$O\left(\binom{|u_1| + \cdots + |u_k|}{|u_1|, \ldots, |u_k|}\right) \text{ pour calculer } \mathop{\sqcup\!\sqcup}_{i=1}^{k} u_i$$

et il construit une représentation du shuffle qui est proportionnelle au nombre de mots dans celui-ci, i.e. en $O\left(|\sqcup\!\sqcup_1^k u_i|\right)$. De plus, cette représentation est facile à utiliser puisqu'il s'agit d'un arbre lexicographique (ou trie).

Même dans le cas plus classique de deux mots u et v, notre algorithme représente une avancée significative. En effet, nous diminuons la complexité d'un facteur $|u| + |v|$ passant de

$$O\left((|u| + |v|)\binom{|u| + |v|}{|u|}\right)$$

pour Spehner [3] à

$$O\left(\binom{|u| + |v|}{|u|}\right).$$

De plus, notre algorithme est beaucoup plus simple à implémenter.

Références

[1] M. Lothaire, editor. *Combinatorics on words.* Cambridge, second edition, 1997.

[2] J. Sakarovitch and I. Simon. Subwords. In Lothaire [1], chapter 6, pages 105–144.

[3] J.-C. Spehner. Le calcul rapide des mélanges de deux mots. *Theoretical Computer Science*, 47 :181–203, 1986.