Génération de chemins de Dyck à pics croissants

Jean-Guy Penaud et Olivier Roques*

LaBRI, Université Bordeaux 1, 351 cours de la Libération, 33405 Talence Cedex, France

Résumé

Le résultat principal de cet article est un algorithme qui engendre de façon aléatoire et uniforme un mot de Dyck à pics croissants en temps quasi-linéaire. Nous montrons que le rapport entre le nombre de mots de Dyck à vallées décroissantes et le nombre de mots de Dyck à pics croissants, de taille fixée, tend vers une constante $c = 2.303727\ldots$ Nous donnons ensuite un algorithme de génération des mots de Dyck à vallées décroissantes en temps quasi-linéaire, en codant ces mots par les mots d'un langage rationnel. Enfin, nous en déduisons un algorithme à rejet pour engendrer les mots de Dyck à pics croissants, avec en moyenne moins de trois échecs.

Abstract

The main result of this paper is an algorithm which generates uniformly at random a Dyck word with increasing peaks, in quasi-linear time. First, we show that the ratio between the number of Dyck words with decreasing valleys and the number of Dyck words with increasing peaks, of a given size, tends to a constant $c = 2.303727\ldots$ Then, we give an algorithm for the generation of Dyck words with decreasing valleys by coding these words with words of a rational language. This leads to a reject algorithm for the generation of Dyck words with increasing peaks, with less than three failures, in average.

Introduction

On s'intéresse à une famille de mots de Dyck avec des contraintes sur les pics, les mots dont la hauteur des pics est croissante, de gauche à droite. Cette contrainte sur les pics rend le langage non algébrique. Intuitivement, on ne peut pas satisfaire cette contrainte avec un automate à pile. Ainsi, les méthodes récursives ([NW78], [Wil77]) de génération, automatisées par Flajolet, Zimmerman et Van Cutsem ([FZC94]) ne sont pas applicables et l'algorithme à rejet que nous proposons est une approche efficace pour résoudre le problème. Dans la deuxième section, nous calculons la série génératrice de ces mots en utilisant une méthode, dite *méthode Temperley.* L'idée principale est de construire à partir d'un mot de Dyck à pics croissants w, un ensemble de mots de Dyck à pics croissants en concaténant à w une pyramide de toutes les façons possibles, de manière à ce que le mot obtenu soit un mot de Dyck à pics croissants. On obtient ainsi une équation, pour la série génératrice, que l'on sait résoudre ([BM96]). On montre ensuite que les coefficients de cette série génératrice croissent exponentiellement comme les coefficients de la série génératrice des mots de Dyck à vallées décroissantes, étudiés dans [BLFP97], et que leurs rapport tend vers $2.303727\ldots$ Ceci va nous permettre de concevoir un algorithme à rejet pour engendrer des mots de Dyck à pics croissants. Remarquons que comme cas particulier de la bijection

*Email: penaud@labri.u-bordeaux.fr, roques@labri.u-bordeaux.fr.

entre les mots de Dyck de longueur $2n$ et les polyominos parallélogramme de périmetre $2n+2$ [DGBV87], les mots de Dyck à pics croissants de longueur $2n$ sont en bijection avec les polyominos parallélogramme de périmetre $2n+2$ dont l'aire des colonnes successives est croissante. Par contre, les mots de Dyck de longueur $2n$ à pics et vallées croissants sont en bijection avec les polyominos parallélogrammes d'aire n, étudiés par Klarner et Rivest ([KR74]). Dans la section 3, nous donnons un algorithme de génération aléatoire et uniforme (`a.e.u`) des mots de Dyck à vallées décroissantes en construisant un automate dont les mots du langage qu'il reconnaît sont en bijection avec les mots de Dyck à vallées décroissantes. Nous montrons que cet automate satisfait certaines propriétés et nous utilisons les résultats de [Den96] pour engendrer de façon aléatoire et uniforme les mots du langage reconnu par l'automate en temps quasi-linéaire. Dans la dernière section, nous donnons une bijection entre les mots de Dyck à pics croissants et les mots de Dyck à vallées décroissantes dont le premier pic est dominant. Ceci conduit à un algorithme à rejet qui engendre de façon aléatoire et uniforme un mots de Dyck à pics croissants en temps quasi-linéaire.

1 Notations et Définitions

Soit X un alphabet et $X^\star$ l'ensemble des mots sur X. Soient $u = u_1 \dots u_k$ et $v = v_1 \dots v_l$ deux mots de $X^\star$, ($u_i, v_j \in X$ pour $1 \leq i \leq k$, $1 \leq j \leq l$), on définit la *concaténation* de u à v par $w = uv = u_1 \dots u_k v_1 \dots v_l$ et u est appelé un *facteur gauche* de w. On note $|w|$ la longueur de w et $|w|_a$ le nombre d'occurrences de la lettre a de X dans w. Soit $X = \{x, \bar{x}\}$, l'ensemble des mots de Dyck est l'ensemble des mots w de $X^\star$ tels que $|w|_x = |w|_{\bar{x}}$ et pour tout facteur gauche u de w, $|u|_x \geq |u|_{\bar{x}}$. Soit $w = w_1 \dots w_{2n}$ un mot de Dyck. Un *pic* dans un mot de Dyck w est un facteur $x\bar{x}$ dans w; une *pyramide* est un facteur $x^i\bar{x}^i$ ($i \geq 1$); une *vallée* est un facteur $\bar{x}x$. Un *chemin* est une suite de points (i, j) de $\mathbb{N} \times \mathbb{N}$. Un *pas* est un couple de deux points consécutifs. Un *pas* **Nord-Est** est un pas $((i, j), (i+1, j+1))$. Un pas **Sud-Est** est un *pas* $((i, j), (i+1, j-1))$. Tout mot de Dyck w de longueur $2n$ peut être représenté par un chemin de Dyck $p = p_0 \dots p_{2n}$ commençant au point $(0, 0)$ et se terminant au point $(2n, 0)$, en associant un *pas* **Nord-Est** à chaque lettre x de w et un *pas* **Sud-Est** à chaque lettre $\bar{x}$ de w.

Un *pic* dans un chemin de Dyck p est un point p_k tel que (p_{k-1}, p_k) est un *pas* **Nord-Est** et (p_k, p_{k+1}) est un *pas* **Sud-Est**.
Une *vallée* dans un chemin de Dyck p est un point p_k tel que (p_{k-1}, p_k) est un *pas* **Sud-Est** et (p_k, p_{k+1}) est un *pas* **Nord-Est**.
Une *pyramide* dans un chemin de Dyck p est une suite de points $(p_0, \dots, p_k, \dots, p_{2k})$ tels que p_k est un *pic* et $(p_0, p_1) \dots (p_{k-1}, p_k)$ sont des *pas* **Nord-Est** et $(p_k, p_{k+1}) \dots (p_{2k-1}, p_{2k})$ sont des *pas* **Sud-Est**.
La *hauteur* h d'un point (i, j) de p est égale à j.
On note $\mathcal{D}$ l'ensemble des mots de Dyck. On note $\mathcal{D}_{pc}$ l'ensemble des mots de Dyck dont la hauteur des *pics* est croissante, de gauche à droite. On note $\mathcal{D}_{vd}$ l'ensemble des mots de Dyck dont la hauteur des *vallées* est décroissante, de gauche à droite. Pour simplifier, on parlera indifféremment de mots ou de chemins.

Définition 1

$$\mathcal{D}_{pc} = \{p \in \mathcal{D} \mid h(p_{k_1}) \leq \dots \leq h(p_{k_l}\} \qquad \textit{où } p_{k_1}, \dots, p_{k_l} \textit{ sont les pics de } p.$$

Définition 2

$$\mathcal{D}_{vd} = \{p \in \mathcal{D} \mid h(p_{k_1}) \geq \dots \geq h(p_{k_l}\} \qquad \textit{où } p_{k_1}, \dots, p_{k_l} \textit{ sont les vallées de } p.$$

2 Série génératrice et asymptotique de $\mathcal{D}_{pc}$

2.1 Série génératrice

Pour trouver la série génératrice des chemins de Dyck dont la hauteur des pics est croissante, nous utilisons une méthode, dite *méthode Temperley*, notamment étudiée dans [BM96] pour l'énumération d'une certaine classe de polyominos. L'idée principale consiste à construire à partir d'un chemin p de $\mathcal{D}_{pc}$, un ensemble $\mathcal{D}_{pc}(p)$ obtenu en concaténant à p, de toutes les façons possibles, une pyramide de hauteur telle que le nouveau chemin obtenu appartienne à $\mathcal{D}_{pc}$ (figure 1).

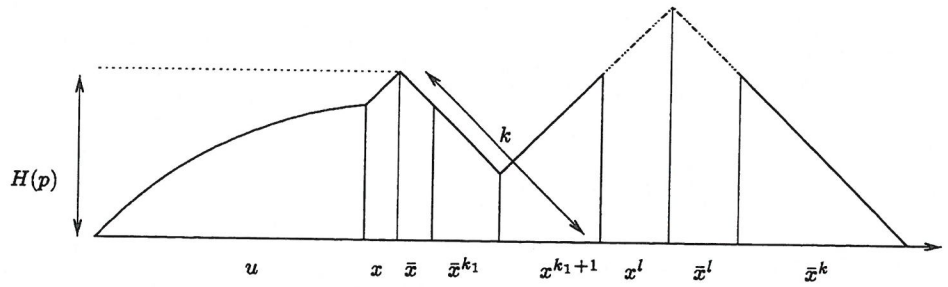

FIG. 1: *Construction de $\mathcal{D}_{pc}(p)$.*

Soit q un chemin de $\mathcal{D}_{pc}(p)$, la construction mentionnée ci-dessus vérifie les propriétés suivantes :

- Si on enlève la dernière pyramide de q, on obtient p.
- La hauteur du dernier pic de q est supérieure ou égale à la hauteur du dernier pic de p.

On remarquera que cette construction donne tous les chemins de $\mathcal{D}_{pc}$ qui ont au moins deux pics.

De façon plus formelle, on définit l'opération $\mathcal{T}$ de $\mathcal{D} \times \mathbb{N}$ dans l'ensemble des parties de $\mathcal{D}$ par :

$$\mathcal{T}(w,l) = \{w' = ux\bar{x}\bar{x}^{k_1}x^{k_1+1}x^l\bar{x}^l\bar{x}^k \mid k > k_1 \geq 0\},$$
$$\text{où } w = ux\bar{x}^k \in \mathcal{D}.$$

Fait : si $w \in \mathcal{D}_{pc}$ alors $\bigcup_{l\geq 0} \mathcal{T}(w,l) = \mathcal{D}_{pc}(w) \subset \mathcal{D}_{pc}$.

Soit $F(s,x)$, la série génératrice bivariée des mots de $\mathcal{D}_{pc}$ où $H(w)$ est la hauteur du dernier pic de w et $L(w)$ est la demi-longueur de w :

$$F(s,x) = \sum_{w\in\mathcal{D}_{pc}} s^{H(w)}x^{L(w)}.$$

La série génératrice des pyramides de hauteur supérieure ou égale à r est

$$\frac{(sx)^r}{1-sx}.$$

$$\begin{aligned} F(s,x) &= \frac{sx}{1-sx} + \sum_{w\in\mathcal{D}_{pc}} s^{H(w)}x^{L(w)} \sum_{k=1}^{H(w)} x^k \frac{1}{1-sx}, \\ &= \frac{sx}{1-sx} + \sum_{w\in\mathcal{D}_{pc}} s^{H(w)}x^{L(w)} \frac{x - x^{H(w)+1}}{(1-x)(1-sx)}, \\ &= \frac{sx}{1-sx} + \frac{x}{(1-x)(1-sx)}\left(F(s,x) - F(sx,x)\right). \end{aligned}$$

Ou encore,

$$F(s,x) = e(s,x) + g(s,x)F(sx,x),$$

avec

$$e(s,x) = \frac{(1-x)sx}{(1-x)(1-sx)-x},$$

et

$$g(s,x) = \frac{-x}{(1-x)(1-sx)-x}.$$

En appliquant la technique de [BM96], on remplace s par sx dans $F(s,x)$ et on obtient une expression de $F(sx,x)$ en fonction de $F(sx^2,x)$. Puis, on remplace la nouvelle expression de $F(sx,x)$ dans $F(s,x)$. En itérant ce procédé à l'infini, on obtient :

$$F(s,x) = \sum_{n\geq 0} \frac{(-1)^n s x^{2n+1}(1-x)}{\prod_{i=1}^{n+1}\left((1-x)(1-sx^i)-x\right)}$$

et en substituant s par 1, on peut établir le théorème suivant :

Théorème 1 *La série génératrice des chemins de Dyck comptés selon la demi-longueur et dont la hauteur des pics est croissante est :*

$$F(1,x) = \sum_{n\geq 0} \frac{(-1)^n x^{2n+1}(1-x)}{\prod_{i=1}^{n+1}\left((1-x)(1-x^i)-x\right)}.$$

2.2 Étude asymptotique

On évalue le comportement asymptotique des coefficients de la série $F(1,x)$. On a :

Théorème 2

$$f_n = [x^n]F(1,x) \sim 0.11997\ldots \left(\frac{3+\sqrt{5}}{2}\right)^n$$

Preuve :
Posons $P_i(x) = (1-x)(1-x^i) - x$ et $H(x) = \sum_{n\geq 0} \frac{(-1)^n x^{2n}}{\prod_{i=2}^{n+1} P_i(x)}$; on a alors :

$$F(1,x) = \frac{x(1-x)}{(1-x\frac{3+\sqrt{5}}{2})(1-x\frac{3-\sqrt{5}}{2})} H(x).$$

L'étude des racines des $P_i(x)$ montre que $H(x)$ converge pour $|x| < \sqrt{2} - 1$. Comme $\frac{3-\sqrt{5}}{2}$ est la plus petite racine de $P_1(x)$ et que $H(\frac{3-\sqrt{5}}{2}) \neq 0$, on peut alors conclure que c'est la singularité dominante de $F(1,x)$. Ceci permet d'écrire

$$f_n \sim K \left(\frac{3+\sqrt{5}}{2}\right)^n$$

et de calculer K par un simple passage à la limite. On trouve alors $K = \frac{\sqrt{5}-3}{\sqrt{5}-5} H(\frac{3-\sqrt{5}}{2}) = 0.11997\ldots$.

3 Chemins de Dyck à vallées décroissantes et génération aléatoire

Il est établi dans [BLFP97] que la série génératrice des chemins de Dyck à vallées décroissantes comptés selon la demi-longueur est rationnelle et satisfait :

$$G(x) = \frac{x(1-x)}{1-3x+x^2}.$$

On en déduit immédiatement que :

$$g_n = [x^n]G(x) \sim \frac{5-\sqrt{5}}{10} \left(\frac{3+\sqrt{5}}{2}\right)^n$$

Il apparaît alors clairement que f_n et g_n croissent exponentiellement de la même façon. Il s'en suit que leur rapport tend vers une constante, ce qui fait l'objet du théorème ci-dessous.

Théorème 3

$$\frac{g_n}{f_n} \sim \frac{5-\sqrt{5}}{10 \times 0.11997\ldots} = 2.303727\ldots$$

Les deux tableaux suivants montrent quelques valeurs de f_n et g_n et leurs rapports.

n	f_n	g_n	n	f_n	g_n
1	1	1	11	5230	10946
2	2	2	12	13464	28657
3	4	5	13	34773	75025
4	9	13	14	90035	196418
5	21	34	15	233590	514229
6	51	89	16	607011	1346269
7	126	233	17	1579438	3524578
8	316	610	18	4114014	9227465
9	800	1597	19	10725109	24157817
10	2040	4181	20	27979704	63245986

n	$\frac{g_n}{f_n}$
10	2.049509
20	2.260423
30	2.295289
40	2.301988
50	2.303359
⋮	⋮
100	2.303727
150	2.303727
300	2.303727

Ce résultat conduit à l'observation suivante : la série génératrice des chemins de Dyck à vallées décroissantes étant rationnelle, il est possible de trouver un algorithme de génération aléatoire et uniforme de ces chemins. D'autre part, supposons que l'on sache associer de façon injective à chaque chemin de Dyck à pics croissants un chemin de Dyck à vallées décroissantes, alors le théorème 3 montre qu'il est tout à fait envisageable en terme de complexité temporelle de concevoir un algorithme à rejet qui engendre `a.e.u` un chemin de $\mathcal{D}_{vd}$, puis construit le chemin de $\mathcal{D}_{pc}$ correspondant s'il existe, et recommence dans le cas contraire. En moyenne, un tel algorithme aura

moins de trois échecs. Pour que cet algorithme soit efficace, il faut que l'algorithme de génération des chemins de $\mathcal{D}_{vd}$ et l'algorithme de construction d'un chemin de $\mathcal{D}_{pc}$ à partir de celui de $\mathcal{D}_{vd}$ soient efficaces.

3.1 Génération

Alain Denise a donné dans [Den96] un procédé systématique de génération aléatoire et uniforme des mots d'un langage rationnel reconnu par un automate dont les séries génératrices des langages associés à chaque état satisfont certaines propriétés. Après avoir énoncé une grammaire d'objets pour les chemins de $\mathcal{D}_{vd}$ ([BLFP97]), nous construirons un automate dont les mots sont en bijection avec les chemins de $\mathcal{D}_{vd}$. Enfin nous montrerons que cet automate satisfait les propriétés requises dans [Den96] et nous expliciterons un algorithme linéaire permettant de construire un chemin de $\mathcal{D}_{vd}$ à partir d'un mot de l'automate.

La figure 2 montre une grammaire d'objets pour les chemins de Dyck à vallées décroissantes. On pourra consulter par exemple [Dut96] pour des détails sur les grammaires d'objets.

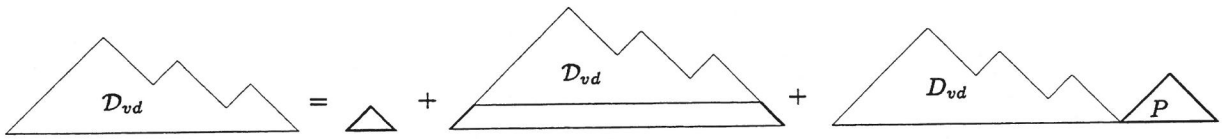

FIG. 2: *Une grammaire d'objets pour* $\mathcal{D}_{vd}$.

Il est suffisant de remarquer que tout chemin de $\mathcal{D}_{vd}$ est soit un chemin *primitif* (*i.e* un chemin ne coupant l'axe origine qu'au premier et dernier point), soit un chemin se terminant par une pyramide, pour prouver que la grammaire de la figure 2 est une grammaire d'objets non ambigüe pour $\mathcal{D}_{vd}$. On en déduit le système d'équations en variables non commutatives de ces chemins.

$$\begin{aligned} \mathcal{D}_{vd} &= x\bar{x} + x\mathcal{D}_{vd}\bar{x} + \mathcal{D}_{vd}P \\ P &= x\bar{x} + xP\bar{x} \end{aligned}$$

La série génératrice $G(x)$ de $\mathcal{D}_{vd}$ est une solution de ce système, en variables commutatives :

$$G(x) = x + xG(x) + \frac{x}{1-x}G(x) = \frac{x(1-x)}{1-3x+x^2}.$$

Bien que $\mathcal{D}_{vd}$ soit un langage algébrique, il est possible de le décrire par un langage rationnel puisque la concaténation d'une pyramide ou d'une semelle (cf. ci-dessous) ont des opérations clairement rationnelles.

La figure 3 montre l'automate $\mathcal{A}$ sur un alphabet à trois lettres $\{a, s, S\}$ qui code les constructions mise en œuvre par la grammaire d'objets.

Un mot reconnu par l'automate s'interprète de la façon suivante : Un a code un facteur $x\bar{x}$. Un S correspond à l'ajout d'une "grande semelle" au mot courant non vide : si à une certaine étape, l'automate a lu un mot u, non vide, alors uS correspond au mot $xu'\bar{x}$ où u' est le mot de $\mathcal{D}_{dv}$ correspondant à u. Un mot as^{k-1} correspond à la concaténation d'une pyramide de taille k au mot courant non vide, (s représentant une "petite semelle" qui surélève un facteur $x\bar{x}$) : si à une certaine étape l'automate a lu un mot u, non vide, alors uas^{k-1} correspond au mot $u'x^k\bar{x}^k$ où u' est le mot de $\mathcal{D}_{dv}$ correspondant à u.

Remarque :
Une pyramide de taille k dans un mot w de $\mathcal{D}_{vd}$ est codée par aS^{k-1} si la pyramide se trouve en tête de w, et par as^{k-1} sinon.

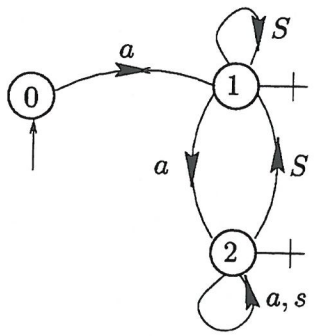

FIG. 3: *L'automate $\mathcal{A}$.*

La proposition suivante est la clé de notre algorithme de génération. Elle s'énonce ainsi :

Proposition 1 *Soient $\mathcal{A} = \langle A, Q, q_o, T, \delta \rangle$ l'automate fini déterministe de la figure 3 avec $A = \{a, s, S\}$, $Q = \{0, 1, 2\}$, $q_0 = \{0\}$, $T = \{1, 2\}$ et $\mathcal{L}(\mathcal{A})$ le langage reconnu par cet automate. Soit $\mathcal{L}^n(\mathcal{A})$ l'ensemble des mots de $\mathcal{L}(\mathcal{A})$ de longueur n et $\mathcal{D}^n_{vd}$ l'ensemble des mots de $\mathcal{D}_{vd}$ de longueur $2n$, alors :*
Les mots de $\mathcal{L}^n(\mathcal{A})$ sont en **bijection** *avec les mots de $\mathcal{D}^n_{vd}$, pour tout n.*

Il est clair que l'automate $\mathcal{A}$ code exactement les constructions mise en œuvre par la grammaire d'objets, ce qui prouve la bijection énoncée à la proposition 1. La figure 4 montre pour $n = 4$ le codage des mots de $\mathcal{D}^n_{vd}$ par les mots de $\mathcal{L}^n(\mathcal{A})$. Nous présentons maintenant un algorithme mettant

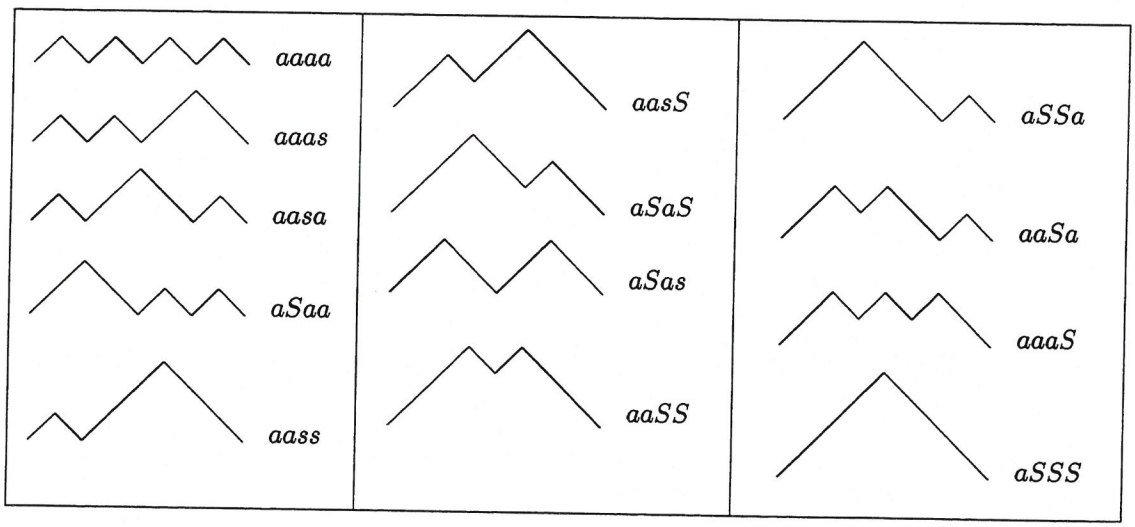

FIG. 4: *Codage des mots de $\mathcal{D}_{vd}$.*

en œuvre la construction d'un mot w' de $\mathcal{D}^n_{vd}$ à partir d'un mot w de $\mathcal{L}^n(\mathcal{A})$. L'algorithme 1 construit w' en partant de la fin de w pour ne pas avoir à redescendre dans w' lorsque un S est rencontré dans w. Le principe est le suivant : lorsqu'un a est rencontré, on ajoute simplement $x\bar{x}$ au début du mot courant. Lorsqu'une suite de *compteur_pyramide* s consécutifs est rencontrée, on écrit au début de w' *compteur_pyramide* $\bar{x}$ consécutifs puis dès qu'un a est rencontré, on écrit $x\bar{x}$ puis *compteur_pyramide* x consécutifs. Enfin chaque fois que l'on rencontre un S, on écrit un $\bar{x}$ au début de w'. Lorsque l'on a fini de lire w, on écrit en début de w', autant de x que l'on a rencontré de S dans w, cette valeur étant mémorisée dans la variable *compteur_hauteur*.

Algorithme 1 Décodage d'un mot de $\mathcal{L}^n(\mathcal{A})$

Entrée $w \in \mathcal{L}^n(\mathcal{A})$
Sortie $w' \in \mathcal{D}_{vd}^n$

```
i ← n
k ← 2n
compteur_pyramide ← 0
compteur_hauteur ← 0
tant que i > 0 faire
  si w(i) = s alors
    w'(k) ← x̄
    compteur_pyramide ← compteur_pyramide + 1
    k ← k − 1 ; i ← i − 1
  sinon
    si w(i) = a alors
      w'(k) ← x̄ ; w'(k − 1) ← x
      k ← k − 2 ; i ← i − 1
      tant que compteur_pyramide > 0 faire
        w'(k) ← x
        k ← k − 1
        compteur_pyramide ← compteur_pyramide − 1
      fin tant que
    sinon
          /* w(i) = S */
      w'(k) ← x̄
      k ← k − 1 ; i ← i − 1
      compteur_hauteur ← compteur_hauteur + 1
    fin si
  fin si
fin tant que
tant que compteur_hauteur > 0 faire
  w'(k) ← x
  k ← k − 1
  compteur_hauteur ← compteur_hauteur − 1
fin tant que
```

La complexité temporelle de l'algorithme 1 est clairement linéaire puisque w et w' ne sont parcourus qu'une seule fois, de droite à gauche. La complexité spatiale est la place nécessaire pour stocker w et w', soit $3n$.

Il nous reste à expliciter l'algorithme de génération aléatoire et uniforme d'un mot de $\mathcal{L}^n(\mathcal{A})$ en utilisant les résultats de [Den96]. D'abord, nous rappelons un algorithme classique de génération des mots d'un langage rationnel, puis on montrera que notre langage satisfait certaines propriétés qui permettent de réaliser de façon efficace l'algorithme de génération, en évitant la manipulation de grand nombres et des calculs coûteux. Ceci est précisément le contenu de [Den96].

Soient $\mathcal{L}$ un langage rationnel et $\mathcal{A} = \langle A, Q, q_0, T, \delta \rangle$ où $A = \{a_1, a_2, \ldots, a_k\}$. Le principe de l'algorithme 2 est de choisir, à chaque étape de la construction du mot w, une lettre à concaténer à w de façon aléatoire et uniforme parmi les lettres possibles, ou de façon équivalente de choisir une lettre a_j avec la probabilité

$$p(i, q, a_j) = \frac{|\mathcal{L}_{q'}^{i-1}|}{|\mathcal{L}_q^i|}$$

où $q' = \delta(q, a_j)$ et $\mathcal{L}_q^i$ est l'ensemble de mots de longueur i reconnus par $\mathcal{A}$ en prenant q comme

Algorithme 2 Génération des mots d'un langage rationnel

Entrée Un automate $\mathcal{A}$ de $\mathcal{L}(\mathcal{A})$, un entier n
Sortie un mot $w \in \mathcal{L}^n(\mathcal{A})$
  $w \leftarrow \epsilon$
  $q \leftarrow q_0$
  tant que $|w| < n$ **faire**
    $a \leftarrow$ une lettre tirée avec la probabilité $p(n - |w|, q, a)$
    $w \leftarrow wa$
    $q \leftarrow \delta(q, a)$
  fin tant que

état initial.

Le choix d'une lettre avec une probabilité donnée se fait avec un algorithme classique, appelé parfois *algorithme d'inversion* (voir par exemple [Dev86]). L'algorithme 3 tire `a.e.u` un nombre

Algorithme 3 Algorithme d'inversion

Entrée Un état $q \in Q$, un entier i
Sortie Une lettre $a \in A$ choisie avec la probabilité $p(i, q, a)$
  $h \leftarrow$ un nombre aléatoire entre 0 et 1
  $j \leftarrow 1$
  $\pi \leftarrow p(i, q, a_j)$
  tant que $\pi < h$ **faire**
    $j \leftarrow j + 1$
    $\pi \leftarrow \pi + p(i, q, a_j)$
  fin tant que
  Retourner a_j

h compris entre 0 et 1 et somme les probabilités jusqu'à majorer h. Un problème apparaît immédiatement : l'algorithme 3 nécessite de calculer $p(i, q, a_j)$ pour tous j tels que $\delta(q, a_j)$ est une transition de l'automate, et donc de manipuler de très grands nombres dès lors que le nombre de mots de taille n d'un langage rationnel croit exponentiellement avec n. C'est le cas de la plupart des langages "intéressants" et c'est précisément le cas de $\mathcal{L}(\mathcal{A})$. Bien heureusement, il existe un moyen d'éviter ces calculs coûteux pour une certaine classe de langages :

Proposition 2 *(Denise [Den96]) Soient $\mathcal{L}$ un langage rationnel et $\mathcal{A} = \langle A, Q, q_0, T, \delta \rangle$ un automate fini déterministe de $\mathcal{L}$. Si pour tout $q \in Q$, il existe un unique pôle de module minimum dans $L_q(t)$, et si ce pôle est simple, alors la probabilité pour que $a \in A$ soit la première lettre d'un mot de $\mathcal{L}_q$ de longueur $n \in N$ engendré par l'algorithme de génération aléatoire s'écrit*

$$p(n, q, a) = \hat{p}(q, a) + \epsilon(n)$$

où $\hat{p}(q, a)$ est un nombre algébrique indépendant de n, et il existe $n_0 \in N, \mu \in [0, 1[$ et un polynôme P tels que pour tout $n \in N, q \in Q, a \in A$,

$$n \geq n_0 \Rightarrow |\epsilon(n)| \leq P(n)\mu^n.$$

On montre facilement que $\mathcal{L}(\mathcal{A})$ satisfait la proposition 2 : Soient $L_0(x), L_1(x), L_2(x)$ les séries génératrices des langages associés à l'automate $\mathcal{A}$ avec respectivement 0, 1 et 2 comme état initial, on a :

$$\begin{aligned} L_0(x) &= xL_1(x), \\ L_1(x) &= 1 + xL_1(x) + xL_2(x), \\ L_2(x) &= 1 + xL_1(x) + 2xL_2(x), \end{aligned}$$

ce qui donne finalement

$$\begin{aligned} L_0(x) &= \frac{x(1-x)}{1-3x+x^2}, \\ L_1(x) &= \frac{1-x}{1-3x+x^2}, \\ L_2(x) &= \frac{1}{1-3x+x^2}. \end{aligned}$$

$L_0(x), L_1(x)$ et $L_2(x)$ ont $1-3x+x^2 = (1-\frac{3+\sqrt{5}}{2}x)(1-\frac{3-\sqrt{5}}{2}x)$ comme dénominateur et $\frac{1}{\frac{3+\sqrt{5}}{2}} = \frac{3-\sqrt{5}}{2}$ est l'unique pôle de module minimum. Ainsi, le langage $\mathcal{L}(\mathcal{A})$ reconnu par l'automate $\mathcal{A}$ de la figure 3 satisfait les conditions de la proposition 2.

L'efficacité de l'algorithme 2 réside exclusivement dans l'efficacité de l'algorithme d'inversion qui peut-être optimisé ainsi : supposons pour simplifier qu'à une étape de l'algorithme, on doit choisir une lettre parmi deux possibles, a_1, a_2 et notons p, la probabilité de choisir la lettre a_1. Supposons que l'on connaisse un intervalle $[p_{min}, p_{max}]$ contenant p. Alors pour choisir la lettre à concaténer au mot courant, on tire un nombre h dans $[0,1]$; si $h < p_{min}$ ou $h > p_{max}$ on choisit respectivement a_1 ou a_2, sinon il faut calculer précisément p pour pouvoir décider. Pour que cette méthode soit efficace, il faut être capable de calculer très rapidement p_{min} et p_{max}, et que la largeur de l'intervalle soit assez petite pour que le calcul exact de p ne s'effectue que très rarement. Ceci est réalisable pour les langages satisfaisant la propriété 2 [Den96], et donc pour notre langage.

Nous énonçons maintenant le théorème [Den96] qui va nous permettre de conclure sur la complexité de l'algorithme de génération des mots de $\mathcal{L}(\mathcal{A})$.

Théorème 4 *(Denise [Den96]) Soient $\mathcal{L}$ un langage rationnel et $\mathcal{A} = \langle A, Q, q_0, T, \delta \rangle$ un automate fini déterministe de $\mathcal{L}$. Si pour tout $q \in Q$, il existe un unique pôle de module minimum dans $L_q(t)$, et si ce pôle est simple. Alors la complexité moyenne en temps de l'algorithme de génération pour générer m mots de $\mathcal{L}$ de longueur n est*

$$\mathcal{F}(n) \sim f(n) + 4mn,$$

où $f(n)$ est une fonction d'ordre polynômial en n. Sa complexité moyenne en espace est

$$\mathcal{M}(m,n) = O(n).$$

La fonction $f(n)$ représente le temps de calcul de n_0, μ et d'une certaine constante K nécessaire à la mise en œuvre de l'algorithme d'inversion efficace. Dans le cas du langage de Fibonacci, Alain Denise a montré dans [Den96] que $f(n) = O(n^2)$. C'est également le cas de notre langage énuméré par les nombres de Fibonacci d'indice impair. On pourra consulter [Knu69] pour des algorithmes numériques faisant intervenir les nombres de Fibonacci et le nombre d'or et qui sont utilisés pour la mise en œuvre efficace de l'algorithme d'inversion.

Nous avons donné plus haut, un algorithme linéaire en temps et en espace pour transformer un mot de $\mathcal{L}^n(\mathcal{A})$ en un mot de $\mathcal{D}^n_{vd}$. On peut donc conclure par la proposition suivante :

Proposition 3 *La complexité temporelle pour engendrer de façon aléatoire et uniforme m mots de Dyck à vallées décroissantes de longueur $2n$ est $O(mn)$ après un pré-traitement de complexité logarithmique en $O(n^2)$.*
La complexité spatiale est $O(n)$.

Remarque :
La complexité *logarithmique* est une mesure de complexité qui suppose qu'une opération arithmétique simple sur un nombre n s'effectue en $O(log(n))$ ([Den96]).

4 Un algorithme à rejet pour la génération des mots de Dyck à pics croissants

Nous avons donné à la section précédente, un algorithme de complexité temporelle quasi-linéaire pour la génération aléatoire et uniforme des mots de Dyck à vallées décroissantes. Nous allons maintenant donner une construction bijective entre les chemins de Dyck à pics croissants et un sous-ensemble des chemins de Dyck à vallées décroissantes, puis nous conclurons par l'algorithme de génération.

Soit p, un chemin de Dyck à pics croissants et $\tilde{p}$ son image miroir par rapport à l'axe horizontal (figure 5). Alors, $\tilde{p}$ est un chemin à vallées décroissantes dont la dernière vallée a la plus petite ordonnée. En coupant $\tilde{p}$ à cette vallée et en recollant le morceau au début du chemin, on obtient un chemin de Dyck à vallées décroissantes dont aucun pic n'est plus haut que le premier pic (à premier pic *dominant*), $\tilde{p'}$. L'opération inverse consistant à couper $\tilde{p'}$ au premier pic et à recoller le morceau à la fin de $\tilde{p'}$, donne $\tilde{p}$ et donc p.

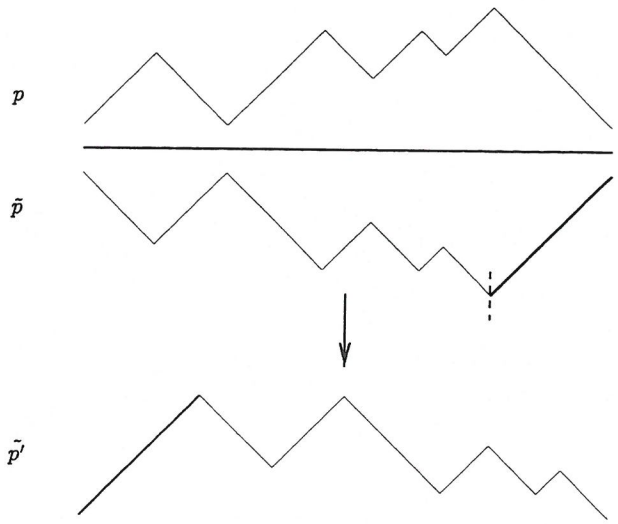

FIG. 5: *Bijection entre les chemin de $\mathcal{D}_{pc}$ et les chemins de $\mathcal{D}_{vd}$ à premier pic dominant.*

Proposition 4 *Les chemins de Dyck à pics croissants de longueur $2n$ sont en bijection avec les chemins de Dyck à vallées décroissantes de longueur $2n$ dont le premier pic est dominant.*

On peut maintenant énoncer l'algorithme de génération aléatoire et uniforme des chemins de Dyck à pics croissants (algorithme 4).

Algorithme 4 Génération `a.e.u` des mots de $\mathcal{D}_{pc}$

Entrée un entier n
Sortie un mot de $\mathcal{D}_{pc}^n$
1: Choisir `a.e.u` un mot w de $\mathcal{D}_{vd}^n$
2: **si** le premier pic de w est dominant **alors**
3: Construire le mot à pics croissants correspondant
4: **sinon**
5: Recommencer
6: **fin si**

Soit w, un mot de $\mathcal{D}_{vd}$, le test qui détermine si le premier pic de w est dominant consiste simplement à parcourir w, à stocker dans une variable la hauteur du premier pic, puis à comparer

cette valeur à la hauteur des autres pics (on s'arrête bien sûr lorsque l'on a trouvé un pic plus haut). Ce test s'effectue en $O(n)$. Remarquons que ce test peut-être effectué directement lors de la génération du mot de $\mathcal{L}(\mathcal{A})$.

La construction du mot à pic croissant peut s'implémenter aisément en temps linéaire en utilisant des opérations binaires élémentaires. Supposons que w soit un mot binaire où 0 code la lettre x et 1 code $\bar{x}$. On compte le nombre k de 0 consécutifs de w en partant du début (de gauche à droite), puis on effectue k décalages à gauche. Enfin on calcule l'image miroir de w en inversant les bits de w.

Après ces observations et le théorème 3 on conclue par la proposition suivante :

Proposition 5 *L'algorithme 4 engendre de façon aléatoire et uniforme m mots de Dyck de longueur $2n$ en $O(mn)$ (complexité temporelle) après un pré-traitement de complexité logarithmique en $O(n^2)$. La complexité spatiale est $O(n)$.*

Remerciements

Nous remercions Mireille Bousquet-Mélou pour de très utiles discussions sur l'analyse asymptotique des séries génératrices.

Références

[BLFP97] E. Barcucci, A. Del Lungo, S. Fezzi, and R. Pinzani. Non-decreasing dyck paths and q-fibonacci numbers. *Discrete Mathematics*, 170:211–217, 1997.

[BM96] M. Bousquet-Melou. A method for the enumeration of various classes of column-convex polygons. *Discrete Mathematics*, 154:1–25, 1996.

[Den96] A. Denise. Génération aléatoire et uniforme de mots de langages ra tionnels. *Theoretical Computer Science*, 159:43–63, 1996.

[Dev86] L. Devroye. *Non-uniform Random Variate Generation.* Springer, Berlin, 1986.

[DGBV87] M.-P. Delest, D. Gouyou-Beauchamps, and B. Vauquelin. Enumeration of parallelogram polyominoes with given bond and site perimeter. *Graphs Combin.*, 3(4):325–339, 1987.

[Dut96] I. Dutour. Grammaires d'objets : énumération, bijections et génération aléatoire. *Thèse de doctorat*, 1996.

[FZC94] Ph. Flajolet, P. Zimmerman, and B. Van Cutsem. A calculus for the random generation of labelled combinatorial structures. In *Theoretical Computer Science, all*, volume 132, pages 1–35. Elsevier Science, 1994.

[Knu69] D.E. Knuth. *The Art of Computer Programming*, volume 2: Seminumerical algorithms. Addison-Wesley Reading MA, 1969.

[KR74] D.A. Klarner and R.L. Rivest. Asymptotic bounds for the number of convex n-ominoes. *Discrete Mathematics*, 8:31–40, 1974.

[NW78] A. Nijenhuis and H.S. Wilf. *Combinatorial Algorithms.* Academic Press, second edition, 1978.

[Wil77] H.S. Wilf. A unified setting for sequencing, ranking, and selection algorithms for combinatorial objects. *Advances in Mathematics*, 24:281–291, 1977.