Tirage aléatoire de mots et d'objets combinatoires

Alain Denise
LABRI
Université de Bordeaux I

8 Février 1993

[résumé par Loÿs Thimonier]

Résumé

On propose ici une généralisation de la méthode florentine [1, 2] de génération aléatoire de certains mots : pour une grande classe, les *fg-langages*, la complexité moyenne de la méthode peut être obtenue simplement au moyen de la série génératrice du langage. L'intérêt de cette approche est son application à l'étude d'objets combinatoires associés par des bijections à de tels langages, en particulier les objets de grande taille : si le nombre des objets de taille n croît exponentiellement, leur génération exhaustive devient vite impossible ; il s'agit alors de pouvoir en temps raisonnable en générer un nombre suffisant pour une étude statistique.

1. Introduction

Soient L_n l'ensemble des mots de longueur n d'un langage L et l_n le cardinal de L_n : la génération aléatoire et uniforme de mots de L consiste, n étant donné à tirer au hasard de façon équiprobable (probabilité : $1/l_n$) un mot de L_n. Cette génération aléatoire est très utile pour l'étude de familles d'objets combinatoires, souvent associées avec des langages par des bijections ; chaque famille admet un paramètre taille, le nombre d'objets de taille $f(n)$ étant égal au nombre l_n de mots de longueur n associés par la bijection ; le nombre d'objets de taille n croît souvent exponentiellement : la génération exhaustive est vite irréalisable, une étude statistique n'est possible que par génération aléatoire et uniforme d'un nombre suffisamment important d'objets de taille n.

Comme les tailles sont grandes, la complexité de la méthode de génération doit être raisonnable en fonction de la longueur des mots. Les algorithmes déterministes permettent le calcul de la complexité dans le pire des cas ; l'algorithme le plus général [3] permet la génération aléatoire et uniforme des mots de tout langage algébrique non ambigu : sa complexité bien que polynomiale fait cependant intervenir l_n, ce qui exclut la génération efficace des très longs mots.

Les méthodes "à tirages" utilisent une suite d'essais-erreurs, entraînant un nombre non borné d'opérations nécessaires pour engendrer un mot : on ne peut prendre en compte qu'une complexité moyenne ; Barcucci, Pinzani et Sprugnoli présentent une telle méthode, avec une complexité linéaire, pour générer les mots de Motzkin ainsi que les chemins sous-diagonaux qui les généralisent [1].

C'est cette méthode qui est généralisée ici à un ensemble plus vaste, les *fg-langages*, avec une complexité moyenne obtenue simplement à l'aide de la série génératrice.

2. Génération de mots

2.1. Méthodes naïve et optimisée : modélisation par des langages. Soient : Random(A) la procédure consistant, dans l'alphabet A de cardinal k, à tirer une lettre de A de façon équiprobable ; ϵ le mot vide ; $|w|$ la longueur de w.

méthode naïve :

entrée : L, n

algorithme :

$w \leftarrow \epsilon$;
tant que $w \notin L_n$:
  $w \leftarrow \epsilon$;
  tant que $|w| \prec n$:
    $a \leftarrow$ Random(A) ;
    $w \leftarrow wa$;

sortie : un mot w de L_n

L'idée pour optimiser [1] est de repérer aussitôt si la construction peut continuer, grâce à un test de "maintien" : le mot en cours doit appartenir alors au langage $FG(L_n)$ des facteurs gauches des mots de L_n ; d'où :

méthode optimisée :

entrée : L, n

algorithme :

$w \leftarrow \epsilon$;
tant que $w \notin L_n$:
  $w \leftarrow \epsilon$;
  tant que $(|w| \prec n) \wedge (w \in FG(L_n))$:
    $a \leftarrow$ Random(A) ;
    $w \leftarrow wa$;

sortie : un mot w de L_n

On remarque que si w ne franchit pas le test de maintien, alors : $w \in R_n$, où R_n est l'ensemble des mots de $FG(L_n)A \setminus FG(L_n)$ de longueur inférieure ou égale à n, donc à la fin le mot ω obtenu après tous les tirages de lettres jusqu'à production d'un mot de L_n appartient à $G \stackrel{def}{=} L_n \cup (\cup_{i \geq 1} R_n^{\,i} L_n)$ vérifiant $G = L_n \cup R_n G$.

2.2. Uniformité de la génération aléatoire des mots de L_n. On doit vérifier qu'un mot v de L_n est généré avec une probabilité $p = 1/l_n$; soit $\sharp L$ le cardinal d'un langage L.

Méthode naïve. $p = \sum_{i>=0} \mathrm{Pr}_i(v)$, où $\mathrm{Pr}_i(v)$ est la probabilité d'obtenir v après la génération de i mots w de $A^n \setminus L_n$: $\mathrm{Pr}(w \in A^n \setminus L_n) = (k^n - l_n)/k^n \Longrightarrow \mathrm{Pr}_i(v) = [(k^n - l_n)/k^n]^i \cdots 1/k^n$, et $p = 1/l_n$.

Méthode optimisée. $p = \sum_{i>=0} \mathrm{Pr}_i(v)$, où $\mathrm{Pr}_i(v)$ est la probabilité d'obtenir v après la génération de i mots w de R_n ; si $R_{n,j} \stackrel{def}{=} R_n \cap A^j$, alors $\mathrm{Pr}(w \in R_n) = \sum_{j=1}^{n} \mathrm{Pr}(w \in R_{n,j}) = \sum_{j=1}^{n} \sharp R_{n,j}/k^j$; de $A^n = L_n \cup (\cup_{j=1}^{n} R_{n,j} A^{n-j})$, on tire : $k^n = l_n + k^n \, \mathrm{Pr}(w \in R_n)$, ce qui permet de terminer avec Pr_i(v) et p comme pour le cas naïf.

2.3. Complexité moyenne.

2.3.1. Rapport avec le nombre moyen l de lettres tirées

Pour la méthode optimisée, le test de maintien est effectué à chaque lettre tirée, alors que par la méthode naïve il n'y a de test que quand on a tiré un multiple de n lettres : le temps de test de maintien doit être négligeable par rapport à celui de la méthode naïve ; on ne considère ici que des langages avec test de maintien en temps constant : la complexité moyenne est proportionnelle au nombre moyen l de lettres tirées pour produire un mot de L_n.

2.3.2. Rapport entre l et le nombre moyen m de mots générés pour obtenir un mot de L_n

LEMME 1. *Pour les 2 méthodes, $m = k^n/l_n$.*

PREUVE. Il s'agit de $E[X]$, où X suit une loi géométrique de paramètre l_n/k^n. Par la méthode naïve, $l = mn$ (chaque mot généré a n lettres) ; par la méthode optimisée, $m - 1 + n < l < mn$ (chacun des $(m-1)$ premiers mots a 1 lettre dans le meilleur des cas, n lettres dans le pire des cas). □

2.3.3. Méthode optimisée : nombre moyen de tirages de lettres pour obtenir un premier mot de L_n et fonctions génératrices

LEMME 2. *Si* $G_i \stackrel{def}{=} \sharp(G \cap A^i)$, $G(t) \stackrel{def}{=} \sum_j G_j t^j$ *alors* $l = (1/k)G'(1/k)$.

PREUVE. Soit X le nombre de lettres à tirer ; alors

$$l = E[X] = \sum_{j>=n} G_j/k^j = (1/k)\sum_j jG_j/k^{j-1} = (1/k)G'(1/k).$$

□

THÉORÈME 1. *Si* $R_{n,j} \stackrel{def}{=} R_n \cap A^j$, $r_{n,j} \stackrel{def}{=} \sharp R_{n,j}$, $R_n(t) \stackrel{def}{=} \sum_{j=1}^n r_{n,j} t^j$, *alors :*

$$l = n + (k^{n-1}/l_n)R_n'(1/k).$$

PREUVE. $G = L_n \cup R_n G \Longrightarrow G(t) = l_n t^n/(1 - R_n(t)$, d'où $P(t) \stackrel{def}{=} G(t/k) = l_n t^n/k^n(1 - R_n(t/k))$; l=P'(1) (lemme) $=nl_n/k^n(1 - R_n(1/k)) + l_n k^{n-1}R_n'(1/k)/k^{2n}(1 - R_n(1/k))^2$;
$A^n = L_n \cup (\cup_{j=1}^n R_{n,j}A^{n-j}) \Longrightarrow k^n = l_n + \sum_{j=1}^n r_{n,j}k^{n-j}$,
d'où $l_n = k^n(1 - R_n(1/k)$: ainsi $l = n + (k^{n-1}/l_n)R_n'(1/k)$. □

3. Fg-langages

3.1. complexité moyenne et fonction génératrice d'un fg-langage L. On souhaite exprimer simplement $R_n'(1/k)$ en fonction de L et n, ce qui va être possible quand L est un fg-langage.

DEFINITION 1. L est *préfixiel* si $FG(L) \subset$ L.

DEFINITION 2. L est *préfixe-complet* si $\forall u \in L, \exists v \neq u, v \in L$, tel que u soit un préfixe (propre) de v.

DEFINITION 3. L est un *fg-langage* s'il est préfixiel et préfixe-complet.

PROPOSITION 1. *Si L est un fg-langage, alors* $\forall u \in L, \forall n \succ |u|, \exists v \in L_n$ *tel que u soit préfixe de v.*

La preuve est aisée.

LEMME 3. *Si L est un fg-langage et* $Q(t) \stackrel{def}{=} R_n(t/k)$, *alors* $Q'(1) = \sum_{i=0}^{n-1} l_i/k^i - nl_n/k^n$.

PREUVE. Avec la proposition précédente, le caractère préfixiel de L, et la définition de R_n. □

LEMME 4. *Si L est un fg-langage,*

$$l = (\sum_{i=0}^{n-1} l_i/k^i)/[t^n]L(t/k).$$

PREUVE. $l = n + (k^n/l_n)(R_n'(1/k)/k)$ (théorème 1) ; L est un fg-langage : $R_n'(t/k)/k = Q'(t)$, et on remplace $R_n'(1/k)/k$ par l'expression de $Q'(1)$ résultant du lemme 3. L'expression obtenue se simplifie en $(\sum_{i=0}^{n-1} l_i/k^i)k^n/l_n$, et $l_n/k^n = [t^n]L(t/k)$. □

THÉORÈME 2. *Si L est un fg-langage, par la méthode optimisée*

$$l = \frac{[t^n](tL(t/k)/(1-t))}{[t^n](L(t/k))}.$$

PREUVE. On manipule une somme double en utilisant la formule du lemme 4. □

3.2. Cas où L est un fg-langage déterministe. Le codage d'objets combinatoires fait souvent intervenir un langage algébrique L ; quand L est en plus déterministe, il est reconnu par états d'acceptation d'un automate à pile déterministe.

THÉORÈME 3. *Si L est un fg-langage algébrique déterministe, alors le test de maintien dans $FG(L)$ peut être effectué en temps constant.*

PREUVE. On parcourt l'automate à pile précédent au fur et à mesure de la construction de w ; $w \in FG(L_n)$ si et seulement si le parcours mène à un état d'acceptation. □

COROLLAIRE 1. *La complexité moyenne de la méthode optimisée pour un fg-langage L algébrique déterministe sur un alphabet A à k lettres est proportionnelle à*

$$\frac{[t^n](tL(t/k)/(1-t))}{[t^n](L(t/k))}.$$

PREUVE. On a déjà vu que pour un langage avec test de maintien en temps constant la complexité moyenne était proportionnelle à l, et on utilise le théorème 2. □

3.3. Applications. On retrouve de façon simple à l'aide d'un logiciel de calcul formel comme ΛΥΩ les résultats de Barcucci, Pinzani et Sprugnoli pour des fg-langages algébriques comme les facteurs gauches de Motzkin et les chemins sous-diagonaux. Les résultats de ce travail sont en cours d'application à d'autres fg-langages (facteurs des suites de Sturm, chemins arborescents, préfixes de Dyck ...).

Bibliographie

[1] Barcucci (E.), Pinzani (R.), et Sprugnoli (R.). – *The Random Generation of Directed Animals.* – Rapport technique n° 11, Lacim, Université du Québec à Montréal, 1992. Actes de l'atelier Franco-Québécois de Combinatoire Algébrique, Eds. P. Leroux et Ch. Reutenauer.

[2] Barcucci (E.), Pinzani (R.), et Sprugnoli (R.). – The random generation of underdiagonal walks. *In : Séries formelles et combinatoire algébrique*, éd. par Leroux (P.) et Reutenauer (C.). pp. 17–32. – Université du Québec à Montréal, 1992. Proceedings of FPSAC'4, Montréal (Canada).

[3] Cohen (Jacques) et Hickey (Timothy). – Uniform random generation of strings in a context-free language. *SIAM Journal on Computing*, vol. 12, n° 4, novembre 1983, pp. 645–655.