Quelques exemples d'algorithmes de génération aléatoire

Dominique Gouyou-Beauchamps
LRI, Université d'Orsay

1er Février 1993

[résumé par Michèle Soria]

Résumé

La génération aléatoire est un outil important pour l'étude expérimentale d'objets combinatoires. Étant donnée la spécification d'une classe de structures combinatoires, il s'agit d'engendrer *uniformément* une structure de taille n (i.e. toutes les structures de taille n ont la même probabilité d'être engendrées).

Cet exposé présente un certain nombre d'algorithmes performants parmi lesquels : les algorithmes de génération d'arbres les plus classiques, l'algorithme de Hickey et Cohen pour générer des mots d'un langage context-free, l'algorithme florentin pour la génération d'animaux dirigés, les algorithmes de L. Alonso pour la génération d'arbres unaire-binaires.

1. Génération de mots de Dyck

Le langage de Dyck sur l'alphabet $\{x, \bar{x}\}$ est engendré par la grammaire $D = \epsilon + xD\bar{x}D$. Les mots de Dyck de taille $2n$ sont en bijection avec les arbres binaires de taille n. Il existe un certain nombre d'algorithmes de génération de mots de Dyck dont le *coût est linéaire* en la taille du mot :

L'algorithme de Rémy. [5] est un algorithme incrémental de génération uniforme d'arbres binaires, fondé sur la relation

$$(n+2)C_{n+1} = 2(2n+1)C_n \ ,$$

où le nombre de Catalan $C_n = \frac{1}{n+1}\binom{2n}{n}$ est le nombre d'arbres binaires de taille n. La construction d'un arbre aléatoire de taille $n+1$ à partir d'un arbre aléatoire de taille n se fait en choisissant une arête (parmi $2n+1$) pour faire l'extension, puis une orientation (gauche ou droite) pour cette arête. Chaque arbre de taille $n+1$ est ainsi obtenu avec multiplicité $n+2$.

L'algorithme de Arnold et Sleep. Il permet d'engendrer efficacement un mot de Dyck aléatoire de taille fixée $2n$. À chaque étape on engendre un x ou un $\bar{x}$ selon un tirage aléatoire dépendant de la fin de mot qu'il reste à construire. Ce tirage est fonction de l'expression du nombre $d_{h,l}$ de suffixes de chemins de Dyck de hauteur initiale h et de taille l :

$$d_{h,l} = \frac{h+1}{l+1}\binom{l+1}{\frac{l+h}{2}+1}.$$

À l'étape i, si le chemin construit est à hauteur h et s'il reste l lettres à engendrer pour former le mot, il suffit de calculer le produit $p = \frac{h+2}{l}\frac{l-h}{2n}$, et l'on engendre x avec probabilité p, et $\bar{x}$ avec probabilité $1-p$.

La factorisation de Raney. Cette factorisation [4] permet d'engendrer des mots de Lukaciewicz. Le langage de Lukaciewicz sur l'alphabet $A = \{x, \bar{x}\}$ est l'ensemble des mots f de A^* tels que $\delta(f) = -1$ et pour tout f' facteur gauche de f, $\delta(f') \geq 0$, (δ est le morphisme défini sur A^* par $\delta(x) = 1$ et $\delta(\bar{x}) = -1$).

D'après le *lemme cyclique* de Raney, tout mot g de A^* vérifiant $\delta(g) = -1$ a une factorisation unique (g_1, g_2) telle que le mot $g_2 g_1$ est dans le langage de Lukaciewicz.

L'algorithme de Raney consiste alors à construire un mot aléatoire formé de n lettres x—pas montants—et $(n+1)$ lettres $\bar{x}$—pas descendants—sans contrainte sur les facteurs gauches, puis à le factoriser par le lemme cyclique (g_2 est le plus grand facteur droit de hauteur minimale).

La factorisation de Catalan. Cette factorisation [4] permet d'engendrer des mots de Dyck à partir de mots formés d'un nombre égal de x et de $\bar{x}$, sans contraintes. Elle reflète la relation

$$(n+1)C_n = \binom{2n}{n}.$$

Soit w un mot ayant même nombre de x et de $\bar{x}$; sa réduction par la relation $x\bar{x} \equiv 1$ donne le mot résiduel $v = \bar{x}^k x^k$. Ayant placé une barre devant le premier x de v (ou au debut du mot si $k = 0$), on échange les lettres résiduelles ($x \leftrightarrow \bar{x}$) dans le mot initial w. Le résultat de cette transformation est un mot de Dyck w' avec une barre devant un $\bar{x}$ ($n+1$ possibilités). Et pour revenir en arrière, on réduit w' par $x\bar{x} \equiv 1$, sans chevaucher la barre, ce qui donne le mot $v' = x^k|\bar{x}^k$, et l'on échange ($x \leftrightarrow \bar{x}$) les lettres de v'.

2. Génération de mots d'un langage algébrique

Hickey et Cohen [3] ont donné une méthode de génération aléatoire des mots d'un langage algébrique défini par une grammaire non ambiguë, sans cycle et sans production vide. Cet algorithme est de complexité $O(n^2 \log^2 n)$, avec un précalcul de complexité $O(n^2 \log n)$ en temps, et $O(n)$ en place.

Étant donnée la grammaire $G = (N, T, A, P)$, où $N = \{N_1, \ldots, N_r\}$ est l'ensemble des non terminaux, T l'ensemble des symboles terminaux, A l'axiome, et P l'ensemble des règles de production $P = \{\pi_{i,j} : N_i \rightarrow \alpha_{i,j} | i = 1, \ldots, r \; ; j = 1, \ldots, s_i\}$, les mots sont engendrés par dérivation gauche.

Soit β un mot de $(N \cup T)^*$ de longueur inférieure à n, et N_i le non terminal le plus à gauche de β. La probabilité de poursuivre la génération en utilisant la règle $\pi_{i,j}$, pour obtenir un mot de longueur n est

$$p_{i,j}(\beta, n) = \frac{g_\gamma(n)}{g_\beta(n)},$$

où γ est le mot dérivé de β par $\pi_{i,j}$, et $g_\delta(n)$ représente le nombre de mots de T^n obtenus par dérivations à partir de δ.

Le noyau de la méthode vient de ce que tout $g_\delta(n)$ s'exprime comme convolution des $g_{N_i}^{(a_i)}(n)$, si le non terminal N_i apparaît a_i fois dans δ.

L'algorithme précalcule les $g_{N_i}(n)(i = 1, \ldots, r)$ en temps $O(n^2 \log n)$: la grammaire étant acyclique, pour chaque n il y a un nombre constant de convolutions à calculer (ce qui se fait en temps $O(n \log n)$ par transformée de Fourier rapide). Ce précalcul nécessite un espace $O(n)$ pour le stockage des $g_{N_i}(n)$.

Le calcul des $g_\delta(n)$ se fait ensuite en temps $O(n \log^2 n)$: $O(\log n)$ pour calculer les puissances des $g_{N_i}(n)$, et $O(n \log n)$ pour calculer les convolutions par FFT.

La génération d'un mot de longueur n, nécessitant $O(n)$ dérivations, se fait donc en $O(n^2 \log^2 n)$ opérations arithmétiques. Et dans le cas des grammaires linéaires, la génération est linéaire, puisqu'il n'y a pas de convolutions.

3. Génération de mots de Motzkin

Le langage des mots de Motzkin sur l'alphabet $B = \{a, x, \bar{x}\}$ est engendré par la grammaire $M = \epsilon + aM + xM\bar{x}M$. Et le langage des facteurs gauches de mots de Motzkin est engendré par la grammaire $F = M + MxF$.

Les mots de Motzkin sont en bijection avec les arbres unaire-binaires, et les facteurs gauches de mots de Motzkin sont en bijection avec les animaux dirigés.

De l'expression des grammaires, on déduit les équations vérifiées par les séries génératrices de dénombrement

$$M(t) = \sum_{n\geq 0} m_n t^n \qquad \text{et} \qquad F(t) = \sum_{n\geq 0} f_n t^n \ ,$$

où m_n (resp. f_n) est le nombre de (facteurs gauches de) mots de Motzkin formés de n lettres :

$$M(t) = 1 + tM(t) + t^2M^2(t) \qquad \text{d'où} \qquad M(t) = \frac{1 - t - \sqrt{(1-3t)(1+t)}}{2t^2} \ ,$$

$$F(t) = M(t) + tM(t)F(t) \qquad \text{d'où} \qquad F(t) = \frac{\sqrt{1+t}}{2t\sqrt{(1-3t)}} - \frac{1}{2t}.$$

La valeur asymptotique des coefficients se déduit par analyse de singularités [6] :

$$m_n \sim \frac{\sqrt{3}}{2\sqrt{\pi}} 3^{n+1} n^{-\frac{3}{2}} \qquad \text{et} \qquad f_n \sim \frac{\sqrt{3}}{\sqrt{\pi}} 3^n n^{-\frac{1}{2}}.$$

Récemment plusieurs algorithmes ont été proposés pour la génération des facteurs gauches de Motzkin d'une part, et des mots de Motzkin d'autre part, avec une complexité moyenne linéaire (même si la complexité maximale est infinie).

3.1. L'algorithme florentin. L'algorithme de Barcucci, Pinzani et Sprugnoli [2] engendre un facteur gauche de Motzkin lettre par lettre, chacune des lettres a, x et $\bar{x}$ apparaissant avec la même probabilité $1/3$. Si au cours de la génération d'un mot on engendre un préfixe qui contient un $\bar{x}$ de plus que de x, on recommence tout le processus. Ainsi chaque mot de B^* a la même probabilité d'être tiré, et donc chaque mot de F apparaît avec la même probabilité $1/f_n$.

Le coût $c(f)$ du tirage d'un mot $f \in F$ est le nombre total de lettres tirées (y compris les échecs) pour engendrer f. Et le coût moyen pour engendrer un préfixe de Motzkin de longueur n est donné par

$$C_n = \frac{1}{f_n} \sum_{f\in B^n} c(f) \ .$$

Or

$$\sum_{f\in B^n} c(f) = \sum_{f\in B^n\cap F} c(f) + \sum_{f\in B^n\cap \bar{F}} c(f)$$

La première somme, coût des essais réussis, est égale à nf_n. La seconde somme, correspondant au coût des essais avec échec, vaut $S_n = \sum_{k=1}^{n} k m_{k-1} 3^{n-k}$, puisqu'un échec à la k-ième lettre implique que l'on avait engendré un mot de Motzkin avec les $(k-1)$ lettres précédentes. Or S_n est le coefficient de t^{n-1} dans $\frac{1}{1-3t}\frac{d}{dt}\big(tM(t)\big)$, qui vaut asymptotiquement $\sqrt{\frac{3}{\pi}} 3^n n^{1/2}$.

Et l'on obtient donc l'expression asymptotique du coût moyen de génération d'un facteur gauche de Motzkin :

$$C_n = \frac{1}{f_n}(nf_n + S_n) \sim 2n.$$

L'algorithme florentin est cependant peu performant pour la génération de mots de Motzkin, puisque son coût moyen est alors quadratique (provenant du terme S_n/m_n).

3.2. Les algorithmes de L. Alonso. L. Alonso a proposé dans sa thèse [1] deux algorithmes pour engendrer des mots de Motzkin avec un coût moyen linéaire. Le premier est fondé sur une subtile méthode d'urnes, et le second est une extension de l'algorithme florentin.

La méthode des urnes. L'algorithme est le suivant ; on répartit les mots de Motzkin de taille n dans v urnes, en plaçant dans l'urne i les N_i mots de Motzkin ayant $i-1$ lettres x. Puis on ajoute dans chacune des urnes un certain nombre de *mots blancs* (mots qui ne sont pas de Motzkin), de façon que l'urne i contienne en tout D_i mots. On choisit alors au hasard l'urne k avec la probabilité $D_k/\sum D_i$ (étape 1). Puis on tire un mot dans l'urne k (étape 2) ; si ce mot n'est pas un mot blanc (*bon choix*) on engendre un mot de Motzkin de taille n ayant k lettres x (étape 3), et sinon on reprend le processus à l'étape 1, jusqu'à ce qu'on fasse un *bon choix*.

Notons $D = \sum_i D_i$ et $N = \sum_i N_i$.

LEMME 1. *La probabilité de faire un bon choix sur l'urne i est $P_i = N_i/N$.*

En effet la probabilité de faire un bon choix sur l'urne i est $\frac{D_i}{D}\frac{N_i}{D_i} = \frac{N_i}{D}$, et la probabilité de faire les étapes 1–2 "pour rien" est $R = 1 - N/D$. Le processus se répétant jusqu'à obtenir un bon choix, on a donc

$$P_i = \frac{N_i}{D}(1 + R + R^2 + \cdots) = \frac{N_i}{N}.$$

LEMME 2. *Le nombre moyen d'étapes 1–2 pour faire un bon choix sur une urne est D/N.*

En effet il y a j étapes 1–2 si les $j-1$ premières sont "pour rien", et la dernière donne un bon choix. Donc le nombre moyen d'étapes 1–2 vaut $\sum_j jR^{j-1}(1-R) = \frac{1}{1-R} = D/N$.

COROLLAIRE 1. *La complexité moyenne de l'algorithme par méthode d'urnes est*

$$\frac{D}{N}(F+G) + H,$$

où F est la complexité moyenne de l'étape 1, G est la complexité moyenne de l'étape 2, et H la complexité moyenne pour construire un mot de Motzkin, étant donné son nombre de x—étape 3.

Il reste donc à déterminer les valeurs de F, G, H et D/N. La complexité moyenne pour construire un mot de Motzkin de taille n ayant k lettres x est linéaire, pour tout k, donc $H = O(n)$. Par ailleurs, un choix judicieux des D_i permet de montrer que F et G sont en $O(n)$, et D/N est en $O(1)$.

Les valeurs de N_i (nombre de mots de Motzkin ayant $i-1$ lettres x) sont connues :

$$N_i = 0 \quad \text{pour} \quad i > 1 + \lfloor n/2 \rfloor, \quad \text{et} \quad N_i = \binom{n}{2i}\frac{1}{i+1}\binom{2i}{i} \quad \text{pour} \quad 1 \le i \le 1 + \lfloor n/2 \rfloor.$$

Les D_i sont choisis légèrement supérieurs aux N_i de façon que la méthode de rejet soit linéaire en moyenne :

$$D_i = \frac{1}{n+1}\binom{n+1-\lfloor\frac{n+1}{3}\rfloor}{i}\binom{n+1}{\lfloor\frac{n+1}{3}\rfloor}.$$

– Pour tout i, N_i/D_i se met sous la forme $\binom{a}{c}/\binom{b}{c} \le 1$.
– L'étape 1, choix de l'urne i avec la probabilité D_i/D peut se faire en engendrant une suite de $n+1-\lfloor\frac{n+1}{3}\rfloor$ bits et en faisant la somme des bits engendrés (et donc $F = O(n)$), puisque

$$\frac{D_i}{D} = \binom{n+1-\lfloor\frac{n+1}{3}\rfloor}{i}/2^{n+1-\lfloor\frac{n+1}{3}\rfloor}.$$

– L'étape 2, validation du choix de l'urne i avec la probabilité N_i/D_i peut se faire en choisissant c entiers dans $[1, b]$ et en vérifiant qu'ils sont tous inférieurs à a, et donc $G = O(n)$.
– L'évaluation asymptotique de D donne $D \sim \frac{1}{2\sqrt{\pi}}3^{n+2}n^{-3/2}$, et donc $D/N \sim \sqrt{3}$.

On obtient donc finalement

THÉORÈME 1. *La complexité de génération aléatoire de mots de Motzkin par méthode d'urnes est en moyenne linéaire.*

Extension de l'algorithme florentin. En utilisant une bijection entre les mots de Motzkin et le sous-ensemble des facteurs gauches de Motzkin qui contiennent au moins un pas horizontal à hauteur 0, L. Alonso montre que l'algorithme florentin peut être étendu à la génération aléatoire de mots de Motzkin avec une complexité moyenne linéaire.

La méthode se déroule en quatre étapes :

– Génération d'un facteur gauche de Motzkin p, de taille $n+1$ qui n'est pas un mot de Motzkin sans pas horizontal à hauteur 0.

– Transformation de p en un facteur gauche p' de hauteur finale impaire.

– Transformation de p' en un mot m, de taille $n+1$ sur $\{x, \bar{x}, a\}^*$, ayant un $\bar{x}$ de plus que de x.

– Transformation de m en un mot de $M\bar{x}$, de taille $n+1$, par application du lemme cyclique.

Le coût moyen de la première étape, en utilisant l'algorithme florentin avec retirage lorsque le facteur gauche tiré est un mot de Motzkin sans pas horizontal à hauteur 0, est linéaire, car la probabilité d'avoir à faire un retirage est en $O(1/n)$. Les deux étapes suivantes sont des transformations bijectives de coût $O(n)$. Enfin l'application du lemme cyclique est aussi en $O(n)$.

L'algorithme est donc de complexité moyenne linéaire, et de plus la génération est uniforme : chaque mot de Motzkin est engendré avec probabilité $1/m_n$.

Bibliographie

[1] Alonso (Laurent). – *Structures arborescentes, algorithmes de génération, problème de l'inclusion, relations maximin.* – Thèse de PhD, Université de Paris-Sud, Orsay, novembre 1992.

[2] 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.

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

[4] Lothaire (M.). – *Combinatorics on Words.* – Addision-Wesley, 1983, *Encyclopedia of Mathematics and its Applications*, volume 17.

[5] Rémy (J.-L.). – Un procédé itératif de dénombrement d'arbres binaires et son application à leur génération aléatoire. *RAIRO Theoretical Informatics and Applications*, vol. 19, n° 2, 1985, pp. 179–195.

[6] Vitter (Jeffrey Scott) et Flajolet (Philippe). – Analysis of algorithms and data structures. *In : Handbook of Theoretical Computer Science*, éd. par van Leeuwen (J.), Chapitre 9, pp. 431–524. – North Holland, 1990.