Projet X-MAP-311-2012-CHAFAI-2

Permutations aléatoires

Proposé par Djalil Chafaï

Second semestre 2011-2012

Beaucoup de situations concrètes nécessitent de permuter aléatoirement une liste finie d'objets. On peut penser par exemple à l'anonymisation mais aussi à la construction de plans d'expériences dans les sciences expérimentales. Ce projet est consacré au problème de la génération de permutations aléatoires sur le groupe symétrique $\mathcal{S}_n$, plus précisément à la simulation de la loi de probabilité uniforme sur $\mathcal{S}_n$.

Loi uniforme sur $\mathcal{S}_n$

Le groupe symétrique $\mathcal{S}_n$ est l'ensemble des permutations de l'ensemble $\{1,\dots,n\}$. Comme il s'agit d'un ensemble fini, la loi de probabilité uniforme μ sur $\mathcal{S}_n$ est définie par $\mu(\{\sigma\}) = 1/\mathrm{card}(\mathcal{S}_n)$ pour tout $\sigma \in \mathcal{S}_n$. Or $\mathrm{Card}(\mathcal{S}_n) = n! \sim \sqrt{2\pi}e^{-n}n^{n+\frac{1}{2}}$, qui est un nombre à environ n chiffres. Cette explosion combinatoire rend difficilement praticable la méthode de simulation habituelle [1] de la loi uniforme sur $\{1,\dots,n\}$, dès que n est grand, qui nécessiterait le stockage et/ou la génération de la liste des éléments de $\mathcal{S}_n$. Nous allons étudier divers algorithmes pour simuler μ, dont celui utilisé par le logiciel Scilab pour implémenter la commande `grand(1,'prm',[1:n])`.

Algorithme naïf

Générons une permutation aléatoire σ en tirant uniformément $\sigma(1)$ dans $\{1,\dots,n\}$, puis $\sigma(2)$ dans $\{1,\dots,n\} \setminus \{\sigma(1)\}$, etc. Il s'agit donc de tirages uniformes sans remise.

Q1 Montrer que σ suit la loi uniforme μ ;

Q2 Préciser la complexité théorique de cet algorithme en fonction de n ;

Q3 Implémenter cet algorithme en langage Scilab ;

Q4 Cet algorithme est-il praticable pour de grandes valeurs de n ?

Q5 Comment générer une permutation de loi uniforme sur $\mathcal{S}_{n+1}$ à partir d'une permutation de loi uniforme sur $\mathcal{S}_n$, en utilisant un seul appel à la fonction `rand` ? Quel est le coût effectif de l'accroissement d'un vecteur par insertion ?

Algorithme de tri

L'idée est de fabriquer une permutation uniforme en réordonnant un désordre symétrique : soit $U_1,\dots,U_n$ des variables aléatoires indépendantes de loi uniforme sur $[0,1]$, et σ une permutation (aléatoire car fonction des U_i) telle que $U_{\sigma(1)} \leq \cdots \leq U_{\sigma(n)}$.

1. En Scilab, `grand(1,1,'uin',1,m)` ou de manière équivalente `ceil(m*rand())`, avec $m = n!$.

Q6 Montrer que σ est unique avec probabilité 1, et que σ suit la loi uniforme μ ;

Q7 Peut-on remplacer la loi uniforme sur $[0,1]$ par une loi quelconque ?

Q8 Peut-on remplacer l'indépendance des U_i par une propriété plus faible ?

Q9 Implémenter cet algorithme en langage Scilab avec des boucles, et comparer avec les performances de l'implémentation grâce à l'usage de la fonction `gsort` ;

Q10 L'aide de Scilab (`help gsort`) mentionne que `gsort` implémente l'algorithme quick-sort. Discuter la complexité de cet algorithme de tri en fonction de n et des données initiales (on pourra consulter les références ainsi qu'Internet).

Algorithme de Fisher-Yates-Knuth

Q11 Soit $\sigma \in \mathcal{S}_n$ et $\tau = (i,j) \in \mathcal{S}_n$ la transposition de deux éléments i, j de $\{1,\dots,n\}$. Montrer que la décomposition en cycles de la permutation $\sigma\tau$ s'obtient à partir de celle de σ en fusionnant deux cycles si i, j appartiennent à des cycles différents de σ, et à fissionner un cycle si i, j appartiennent au même cycle ;

Q12 Soit $(\sigma_n)_{n\geq 0}$ la suite récurrente aléatoire (connue sous le nom de processus de restaurants chinois), avec σ_n à valeurs dans $\mathcal{S}_n$ pour tout $n \geq 1$, construite de la manière suivante : $\sigma_1 = (1)$, et pour tout $n \geq 1$, σ_{n+1} est construite à partir de σ_n en créant un nouveau cycle de longueur 1 avec probabilité $1/(n+1)$, ou en rejoignant uniformément le cycle ς de σ avec probabilité $\mathrm{card}(\varsigma)/(n+1)$. Montrer que pour tout $n \geq 1$, la permutation aléatoire σ_n suit la loi uniforme μ sur $\mathcal{S}_n$;

Q13 Déduire des deux questions précédentes que si $U_1,\dots,U_n$ sont des variables discrètes indépendantes avec U_i de loi uniforme sur $\{1,\dots,i\}$ pour tout $1 \leq i \leq n$, alors la permutation aléatoire $(1,U_1)\cdots(n,U_n)$ suit la loi uniforme μ sur $\mathcal{S}_n$;

Q14 En déduire un algorithme de simulation de μ basé sur un boucle `for` et $n-1$ appels à la fonction `rand()`. Implémenter cet algorithme en langage Scilab, et comparer ses performances avec un simple appel à `grand(1,'prm',[1:n])` ;

Q15 Il se trouve que `grand(1,'prm',[1:n])` implémente cet algorithme. Comment expliquer les différences de performances observées précédemment ?

Phénomène de convergence abrupte de Diaconis-Shahshahani

Le groupe symétrique $\mathcal{S}_n$ comprend $\binom{n}{2}$ transpositions non triviales. On appelle ici *loi uniforme sur les transposition* la loi ν sur $\mathcal{S}_n$ donnée pour tout $\sigma \in \mathcal{S}_n$ par

$$\nu(\{\sigma\}) = \begin{cases} 1/n & \text{si } \sigma \text{ est l'identité ;} \\ 2/n^2 & \text{si } \sigma \text{ est une transposition ;} \\ 0 & \text{sinon.} \end{cases}$$

Soit $(T_k)_{k\geq 1}$ une suite de variables aléatoires sur $\mathcal{S}_n$, indépendantes et de même loi ν. Pour tout $k \geq 1$, le produit de transpositions aléatoires $T_1 \cdots T_k$ a pour loi ν^{*k}, où la convolution est celle du groupe $\mathcal{S}_n$ (non abélien !). La proximité de ν^{*k} et μ peut être quantifiée au moyen de la distance en variation, définie par

$$d_{n,k} := \sum_{\sigma\in\mathcal{S}_n} \left| \mathbb{P}(T_1\cdots T_k = \sigma) - \frac{1}{n!} \right| = \sum_{\sigma\in\mathcal{S}_n} \left| \nu^{*k}(\{\sigma\}) - \mu(\{\sigma\}) \right| = 2 \max_{A\subset\mathcal{S}_n} \left| \nu^{*k}(A) - \mu(A) \right|.$$

Diaconis et Shahshahani ont établit que si $n \geq 10$ alors

$$d_{n,k} \leq c^{\text{te}} e^{-\frac{2k}{n}+\log(n)}$$

tandis que pour tout $k \geq 1$,

$$d_{n,k} \geq 2\left(\frac{1}{e} - e^{-e^{-\frac{2k}{n}+\log(n)}}\right) + o_{n\to\infty}(1).$$

Ainsi, $d_{n,k}$ se rapproche abruptement de 0 lorsque k dépasse $\frac{1}{2}n\log(n)$, comme l'avait conjecturé Aldous. Ce phénomène de convergence abrupte a lieu pour de nombreuses autres suites récurrentes aléatoires, comme le montre la lecture du livre beaucoup plus récent de Levin, Peres, et Wilmers. La preuve originelle de Diaconis et Shahshahani utilise un ensemble test A bien choisi pour la borne inférieure, et la théorie des représentations (groupe symétrique) pour la borne supérieure.

Q16 On pourra illustrer le résultat de Diaconis et Shahshahani avec le logiciel Scilab.

Algorithme de Steinhaus-Johnson-Trotter

L'algorithme de Steinhaus-Johnson-Trotter est un algorithme déterministe qui permet de parcourir tous les éléments de $\mathcal{S}_n$ sans avoir besoin de comparer aux éléments déjà produits. Algébriquement, il correspond à parcourir un graphe de Cayley de $\mathcal{S}_n$, tandis que géométriquement, il correspond à parcourir les sommets adjacents du polytope appelé permutaèdre. Plus précisément, on fabrique une suite récurrente $(\sigma_k)_{1\leq k\leq n!}$ sur $\mathcal{S}_n$ en construisant σ_{k+1} à partir de σ_k comme suit :

1. pour toute position $i \in \{1,\dots,n\}$, si $\sigma_k(1),\dots,\sigma_k(i-1)$ correspond à une permutation paire, on la marque 1, et on la marque 0 sinon ;
2. s'il existe une plus grande position $r \in \{1,\dots,n\}$ marquée 1 et possédant une position marquée 1 immédiatement à sa gauche, alors on permute les valeurs associées de σ_k, on incrémente k, et on boucle à l'étape précédente, sinon l'algorithme est terminé.

Q17 Évaluer la complexité de l'algorithme.

La structure récursive sur n de l'algorithme est la suivante : les éléments de $\mathcal{S}_n$ s'obtiennent à partir de ceux de $\mathcal{S}_{n-1}$ en insérant n à chaque position possible dans chacun des éléments de $\mathcal{S}_{n-1}$. Si l'élément de $\mathcal{S}_{n-1}$ est pair, alors n est inséré dans toutes les positions possibles par ordre décroissant, et sinon par ordre croissant.

Le livre de Knuth donne de nombreux algorithmes alternatifs à celui-ci.

Références

- Persi Diaconis, Mehrdad Shahshahani, *Generating a random permutation with random transpositions*, Probability Theory and Related Fields, vol. 57(2), 159–179 (1981)
- Rajeev Motwani, Prabhakar Raghavan, *Randomized Algorithms*, CUP (1995)
- Donald Knuth, *The Art of Computer Programming vol. 2 et 4*, Addison–Wesley
- David A. Levin, Yuval Peres, Elizabeth L. Wilmer, *Markov Chains and Mixing Times*, American Mathematical Society (2008)