Introduction aux générateurs de nombres pseudo-aléatoires en informatique

Pierre Callebut

Août 2012

Résumé

Ce document a pour objectif de présenter de manière succincte aux étudiants en informatique la problématique de la génération de nombres aléatoires sur ordinateur. Après une brève introduction pour justifier l'intérêt pratique du sujet, le concept de générateur de nombres pseudo-aléatoires est présenté. La qualité du générateur, la sélection de la graine et quelques exemples historiques sont abordés. Les références et les exemples fournis ont pour objectif de permettre à tout étudiant, après une lecture critique de l'article, de s'initier aux générateurs de nombres pseudo-aléatoires et à leur conception.

De nombreuses applications nécessitent aujourd'hui la génération de nombres aléatoires. Les jeux électroniques, par exemple, utilisent généralement des valeurs aléatoires pour créer un scénario nouveau à chaque lancement. Les nombres aléatoires sont également indispensables dans le domaine de la sécurité informatique, pour générer par exemple des mots de passe. Ils sont aussi primordiaux pour faire communiquer des ordinateurs sur des réseaux locaux : lorsque deux ordinateurs utilisent le canal de communication en même temps et créent une collision, ils attendent un temps aléatoire avant de recommencer à émettre. Même l'utilisation d'une télécommande de voiture, d'une carte de transport public, d'un téléphone mobile, d'un réseau WiFi, ou d'une souris sans fil nécessite l'utilisation de nombres aléatoires.

Générer des nombres aléatoires est fondamental mais cette opération est délicate, même impossible avec un ordinateur classique. Comment en effet générer des nombres aléatoires à partir d'une machine aussi déterministe ?

À défaut de pouvoir générer des nombres aléatoires, les ordinateurs classiques doivent se contenter de nombres dits *pseudo-aléatoires*. En termes vulgarisés, un *générateur de nombres pseudo-aléatoires* (PRNG [1]) est un algorithme qui génère une séquence de nombres qui *ressemble* à une séquence de nombres tirés au hasard. L'algorithme exécuté sur un ordinateur est déterministe : il prend en entrée un nombre aléatoire, appelé *graine* et produit à chaque itération un nouveau nombre qui est seulement *pseudo-aléatoire*. Le nombre d'états dans un ordinateur étant fini, la séquence de nombres pseudo-aléatoires générés entre nécessairement dans un cycle à partir d'un certain nombre d'itérations. La longueur de ce cycle est appelée la *période* du générateur.

1. *Pseudo-random Number Generator*

Exemple 1 (Qualité du générateur) *Intuitivement, un générateur qui simule des lancers de dés sera considéré comme de piètre qualité si la séquence générée est "1, 1, 1, 1,..." ou encore "1, 2, 3, 4, 5, 6, 1, 2, 3, 4, 5, 6,...". En revanche, la séquence "3, 4, 4, 4, 5, 6, 2, 1, 3, 1,..." pourrait avoir été générée par un générateur de bonne qualité, mais la longueur de la séquence est trop courte pour en être vraiment convaincu.*

Il n'existe malheureusement pas de méthode infaillible pour évaluer la qualité d'un PRNG. L'évaluation se fait généralement par le biais d'une analyse mathématique, en particulier statistique. Les deux propriétés majeures qui doivent être assurées sont : (a) la séquence générée doit être statistiquement indistinguable d'une séquence aléatoire, (b) les nombres générés ne doivent pas être prédictibles. Il est important de souligner que ces deux propriétés doivent être vérifiées quelle que soit la valeur de la graine choisie. Des outils sont gratuitement disponibles à partir du site web [2] du NIST [3] pour réaliser des batteries de tests statistiques. Ces tests ne sont pas suffisants pour être convaincu de la qualité d'un PRNG, mais ils constituent une première étape incontournable.

La génération de la graine est une opération délicate car cette dernière doit être aussi aléatoire que possible. Demander à l'utilisateur de jeter un dé plusieurs fois permettrait de générer une graine de bonne qualité, mais cette procédure n'est pas réellement compatible avec les contraintes ergonomiques des applications usuelles. En pratique, la graine est générée à partir d'un phénomène physique difficilement maîtrisable. Utiliser seulement l'horloge de l'ordinateur n'est clairement pas satisfaisant. En revanche, utiliser les mouvements de la souris, les accès à la mémoire ou, mieux, le bruit généré par les microcircuits sont des approches bien meilleures pour générer une séquence de nombres ressemblant à une séquence aléatoire. L'algorithme qui produit une séquence de cette manière est appelé un *générateur physique*, noté HRNG [4] ou TRNG [5]. Dans les systèmes d'exploitation de type Unix, il existe un HRNG qui produit en permanence des nombres "aléatoires" et les stocke dans le fichier `/dev/random`.

Si un HRNG produit des nombres aléatoires de meilleure qualité qu'un PRNG, il pourrait être tentant de penser que les PRNG sont inutiles. Ce n'est pas le cas. Les HRNG présentent en effet deux inconvénients majeurs. Tout d'abord, ces générateurs qui reposent sur des phénomènes physiques sont très lents, car les événements qui ont lieu sur un ordinateur produise peu d'*entropie*. Ensuite, la séquence produite par un HRNG n'est pas reproductible. Cela peut être une propriété désirable, mais cela peut aussi constituer un handicap pour certaines applications, par exemple pour rejouer un même scénario dans un jeu.

L'évolution des outils de chiffrement à partir de la fin du XIXè siècle a rapidement montré la nécessité de générer de manière mécanique des nombres pseudo-aléatoires. C'est toutefois l'émergence de l'informatique qui a véritablement lancé la recherche sur les PRNG, avec les travaux de John von Neumann au lendemain de la seconde guerre mondiale. Il proposa notamment le générateur *middle-square*, utilisé dans l'ENIAC [6]. Le générateur middle-square est très simple. Il consiste à prendre un nombre de n chiffres, qui constitue la graine,

2. `http://csrc.nist.gov/groups/ST/toolkit/rng/batteries_stats_test.html`
3. *National Institute of Standards and Technology*
4. *Hardware Random Number Generator*
5. *True Random Number Generator*
6. *Electronic Numerical Integrator And Computer*

l'élever au carré et ne garder que les n chiffres du milieu, qui représentent le nombre pseudo-aléatoire généré et sur lequel l'algorithme est réitéré.

Exemple 2 (Générateur middle-square) *On considère une graine* 9999 *que l'on élève au carré :* $9999^2 = 99980001$*. Le nombre généré est alors* 9800 *qui est ensuite réutilisé pour générer le nombre suivant. On obtient ainsi la séquence :* 9800*,* 0400*,* 6000*,* 0000*,* 0000*,* 0000*, etc. Dans cet exemple où* $n = 4$ *et où la graine est* 9999*, la période du générateur est 1.*

Dès 1949, Dirk Lehmer propose à son tour les *générateurs congruentiels linéaires* [2], qui sont encore utilisés aujourd'hui, bien que la qualité de la séquence générée ne soit pas toujours satisfaisante. Le générateur linéaire congruentiel est de la forme $X_{n+1} \equiv aX_n + c \pmod{m}$ où les X_n $(n > 0)$ sont les nombres pseudo-aléatoires générés, X_0 est la graine, et a, c et m sont les paramètres du générateur. Autrement dit, X_{n+1} est le reste de la division entière de $(aX_n + c)$ par m. Choisir les paramètres du générateur n'est pas trivial, comme l'explique Donald Knuth dans [1].

Exemple 3 (Générateur linéaire congruentiel) *On considère le générateur* $X_{n+1} \equiv 5X_n \pmod{13}$ *avec la graine* $X_0 = 11$*. La séquence générée est alors* 3*,* 2*,* 10 *et* 11*. La période de ce générateur avec la graine considérée est 4.*

D'autres générateurs de nombres pseudo-aléatoires ont depuis lors vu le jour, reposant sur des fonctions mathématiques, par exemple le générateur Blum–Blum–Shub ou les générateurs de type Fibonacci, ou reposant sur des constructions cryptographiques. Le choix du générateur dépend des contraintes de l'application, c'est-à-dire principalement des ressources disponibles sur l'appareil destiné à utiliser le générateur, mais aussi de l'exigence sur la qualité de la séquence générée.

Références

[1] Donald E. Knuth. *The Art of Computer Programming. Volume 2 : Seminumerical Algorithms.* Addison-Wesley, 1969.

[2] Derrick H. Lehmer. Mathematical methods in large-scale computing units. In *Proceedings of a Second Symposium on Large-Scale Digital Calculating Machinery, 1949*, pages 141–146, Cambridge, Massachusetts, USA, 1951. Harvard University Press.