Aléas du hasard informatique

JEAN-PAUL DELAHAYE

Le vrai hasard étant hasardeux, contentons-nous d'un pseudo-hasard et adaptons-le à nos besoins.

Un calcul composé de milliards d'étapes successives peut être répété et donner exactement le même résultat : le hasard a été éliminé des systèmes informatiques. Cette extraordinaire maîtrise des petits courants électriques, que l'on combine pour exécuter un programme, est époustouflante. Dans nos ordinateurs, le hasard est maîtrisé jusqu'au niveau microscopique : il a disparu ! N'est-il pas étonnant qu'on en ait besoin ? Après avoir tout fait pour l'éliminer, il faut recréer le hasard...

Roland Moreno, plus connu pour son invention de la carte à puce, avait perfectionné, dans sa jeunesse journalistique, un dispositif automatique pour tirer à pile ou face. L'informatique supplée au tirage physique dans un grand nombre d'applications où il faut engendrer rapidement des chiffres aléatoires.

La difficulté est la suivante : il faut imiter autant que possible un tirage à pile ou face, où la connaissance d'un lancer ne permet pas de déterminer le résultat du lancer suivant, et ce, par un programme fondé sur une formule mathématique d'utilisation simple par l'ordinateur. Or, qui dit formule mathématique, dit déterminisme, donc un hasard limité.

Nécessité du hasard

Le hasard est nécessaire dans les logiciels de *simulation* : pour reproduire des phénomènes aléatoires, il faut disposer de briques élémentaires d'*aléas*, c'est-à-dire de nombres aléatoires.

En *cryptographie* aussi, où, pour échapper aux espions, on doit déguiser les messages en les combinant avec des séquences de chiffres quelconques, c'est-à-dire aléatoires : on constate d'ailleurs qu'à partir d'un bon système de codage, on obtient un bon système de création de suites aléatoires et... inversement.

Le hasard est nécessaire en *théorie de l'information*, où les séquences au plus fort contenu en information sont justement les séquences vraiment aléatoires. Aussi s'est développée une science de la production contrôlée de l'aléa, qui a aujourd'hui atteint un grand raffinement.

En réalité, il y a au moins trois disciplines dans la science du hasard : celle du *hasard faible,* utile pour les simulations, celle du *hasard moyen,* qui convient en cryptographie, et celle du *hasard fort,* née de la théorie de l'information et de la calculabilité.

Le hasard faible

Pour créer un nuage de points noirs destinés à colorier en gris une partie d'image (et plus généralement dans

TESTS SPECTRAUX : LES RÉSEAUX DE POINTS

Les générateurs congruentiels linéaires sont utilisés pour définir les fonctions *random* des langages de programmation. On se donne M, a et b, ainsi qu'un point de départ appelé *graine* $x(0)$ et L'on calcule $x(1)$, $x(2)$, etc., avec la formule :

$x(n+1) = (a\,x(n) + b) \mod M \qquad u(n) = x(n)/M$

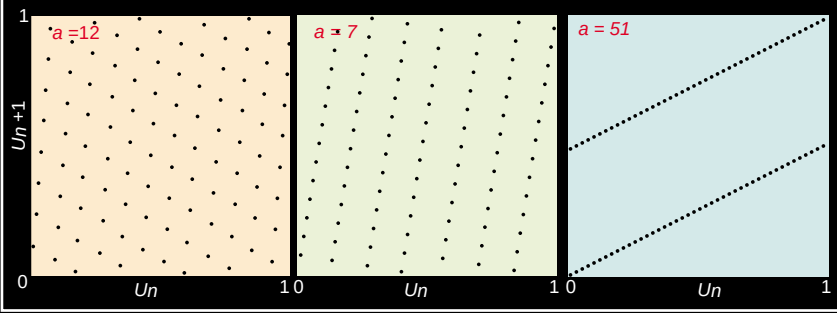
($z \mod M$ désigne le reste de la division de z par M)

Pour en tester les qualités, en plus de la période qu'on s'arrange pour obtenir la plus grande possible, on associe à ces suites des points dans un espace de dimension deux ou plus. On obtient un réseau régulier dont les caractéristiques donnent des indications sur la qualité du générateur pseudo-aléatoire.

Exemple. Avec $M = 101$, $b = 0$. On représente les points de coordonnées $u(n)$, $u(n+1)$ pour trois valeurs de a , $a = 12$, $a = 7$, $a = 51$, $b = 0$.

La période est à chaque fois maximale, c'est-à-dire 100.

On voit très clairement la structure en réseau.



1. En prenant une suite pseudo-aléatoire de nombre entre 0 et 1, $u(0)$, $u(1)$, etc. et en dessinant les points $M(0) = (u(0), u(1))$, $M(1) = (u(1), u(2))$, etc., on obtient une figure où l'imperfection des $u(i)$ apparaît instantanément. Ici on examine les générateurs congruentiels linéaires. Pour $a = 12$, les points sont bien distribués (trop bien ?). Pour $a = 7$, le résultat est un peu moins bon, et pour $a = 51$, il n'est pas satisfaisant. Des méthodes mathématiques et des algorithmes spécialisés permettent de calculer les meilleurs jeux de paramètres pour que les réseaux (en dimension deux ou plus) soient satisfaisants. Un générateur de ce type, nommé RANDU, fut exploité pendant des années dans toutes sortes de logiciels (en particulier, dans le système d'exploitation des machines de la fameuse gamme IBM 360), avant qu'on ne découvre ses graves insuffisances. Les paramètres utilisés étaient : $M = 2^{31}$, $a = 65539$, $b = 0$.

tout problème de texture), pour simuler la circulation automobile d'un réseau autoroutier, pour étudier des molécules de gaz dans un réservoir ou encore des interactions entre particules élémentaires, il faut des méthodes rapides produisant des millions de bits aléatoires, même s'ils ne le sont que superficiellement : un hasard faible suffit.

George Marsaglia, de l'Université de Floride, spécialiste des générateurs aléatoires, pense que dans bien des simulations même un médiocre hasard convient : «Un générateur aléatoire, écrit-il, c'est comme le sexe : quand c'est bon c'est merveilleux, et quand c'est mauvais c'est encore assez bon.»

Les fonctions *random* des langages de programmation procurent ce hasard peu coûteux en temps de calcul. On les utilise assez souvent avec satisfaction, mais mille exemples ont montré, dans le passé, qu'en cas d'expérimentations intensives ou cruciales de désagréables surprises se produisent. En 1993, des chercheurs de l'Université de Géorgie, explorant, par simulation, un problème de physique des solides, obtinrent des résultats en contradiction avec ceux obtenus par calculs directs. Une autre équipe découvrit un an plus tard que ces comportements aberrants étaient dus à l'utilisation de générateurs aléatoires insuffisants : le risque n'est pas très élevé en simulation, mais si l'on veut s'en garder, alors les difficultés commencent, et cette fois la règle est que, pour obtenir un hasard garanti, il ne faut surtout pas faire n'importe quoi.

LES GÉNÉRATEURS LES PLUS SIMPLES

Les générateurs congruentiels linéaires sont souvent utilisés pour les fonctions *random*. Leur mode de fonctionnement est le plus simple possible, et donc rapide.

On part de trois nombres entiers : *M* qui fixe l'intervalle où les nombres de la suite vont être engendrés, *a* et *b* qui sont pris entre *0* et *M* – 1. On choisit un quatrième nombre, la *graine x(0)*, elle aussi entre 0 et *M* – 1. On changera cette graine si l'on veut, avec le même générateur, obtenir une autre suite, ou on la reprendra si l'on souhaite obtenir exactement le même «hasard» lors d'un second essai, ce qui est souvent utile pour la mise au point d'un programme. La suite pseudo-aléatoire est donnée par les calculs suivants :

x(1)=reste de la division de *ax(0)+b* par *M*
x(2)=reste de la division de *ax(1)+b* par *M*
etc.

Chaque *x(i)* est un nombre entier compris entre 0 et *M* – 1, qu'on utilise pour engendrer des bits aléatoires (0 ou 1) ; ce choix est fait, par exemple, en prenant la parité de *x(i)*, – 0 si *x(i)* est pair, et 1 si *x(i)* impair – (ou mieux, en calculant la parité de la somme des chiffres de *x(i)*, 0 si cette somme est paire, 1 si cette somme est impaire).

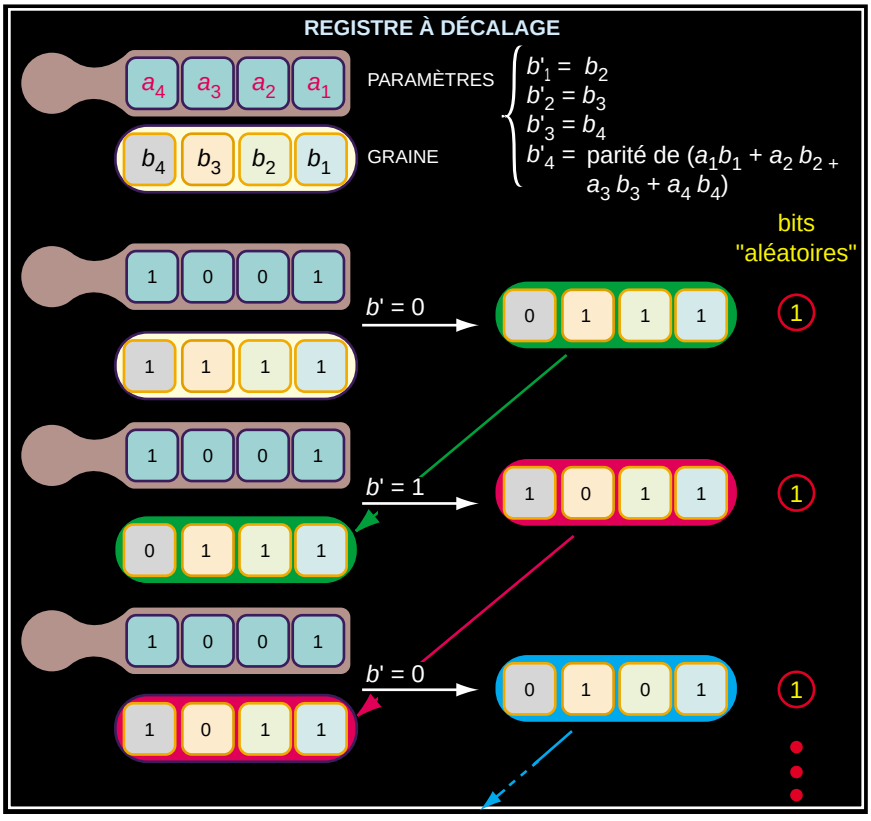
On peut transformer les entiers *x(i)* pour donner des nombres réels. En prenant par exemple $u(i) = x(i)/M$, on obtient une suite de nombres réels *u(i)* compris entre 0 et 1, dont la distribution est, en première approximation, uniforme : chaque sous-intervalle de [*y*, *z*] de [0, 1] contient *x(i)* un nombre de fois proportionnel à sa longueur *z* – *y*.

Les paramètres suivants ont été utilisés : $M = 2^{31} - 1$, $a = 16807$, $b = 0$.
Avec $x(0) = 12345$, on obtient :
$x(1) = 207482415$ $u(1) = 0{,}0966165285$
$x(2) = 1790989824$ $u(2) = 0{,}8339946274$
$x(3) = 2035175616$ $u(3) = 0{,}9477024977$
etc.

La question est : comment obtenir des garanties sur le caractère aléatoire de la suite ? Persuadons-nous d'abord qu'une suite engendrée par le générateur précédent finira par tourner en rond. En effet, il n'y a qu'un nombre fini de valeurs possibles pour *x(i)* (les entiers entre 0 et *M* – 1), et donc quand apparaît une valeur déjà obtenue – ce qui se produit nécessairement – la suite repasse par les mêmes valeurs, dans le même ordre, et ce, indéfiniment. Le nombre d'étapes pour revenir aux mêmes calculs est la période.

Cette ronde n'est pas si grave qu'elle en a l'air, et tout générateur fondé sur une formule du type précédent présentera ce défaut. Essayons d'apprivoiser cette ronde.

Il est clair qu'on ne saurait se satisfaire, pour la simulation, d'une suite dont la période serait trop petite. On souhaite que la suite ait une grande période, voire la plus grande possible, c'est-à-dire *M*. L'analyse mathématique du problème a conduit au résultat suivant, démontré au début des années 1960 : si les nombres *b* et *M* n'ont pas de diviseurs communs, si tout nombre premier qui



2. Le registre à décalage (à rétroaction linéaire) est défini par le mécanisme suivant. Une série de *N* bits est fixée a_1, a_2, a_3, a_4. Ce sont les paramètres, ici, 1 0 0 1. On initialise le générateur avec quatre bits b_1, b_2, b_3, b_4. (Ils constituent la *graine*. Ici, 1 1 1 1). À chaque étape, un nouveau bit *b'* est introduit : *b'* = 0 si (a_1 b_1 + a_2 b_2 + a_3 b_3 + a_4 b_4) est pair, 1 sinon (dans notre exemple, il suffit de compter le nombre de 1 parmi les bits b_1 et b_4 : s'il y en a 0 ou 2, *b'* vaut 0, sinon *b'* vaut 1). Le nouveau bit *b'* remplace b_4, lequel remplace b_3, lequel remplace b_2, lequel remplace b_1. Le bit b_1, lui, est utilisé comme bit aléatoire et disparaît du quadruplet. Dans notre exemple, on obtient successivement les quadruplets : 1111 0111 1011 0101 1010 1101 0110 0011 1001 0100 0010 0001 1000 1100 1110, puis la suite se répète. Les chiffres «aléatoires» engendrés sont obtenus en prenant le dernier de chaque série de quatre : 111101011001000...