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 :

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

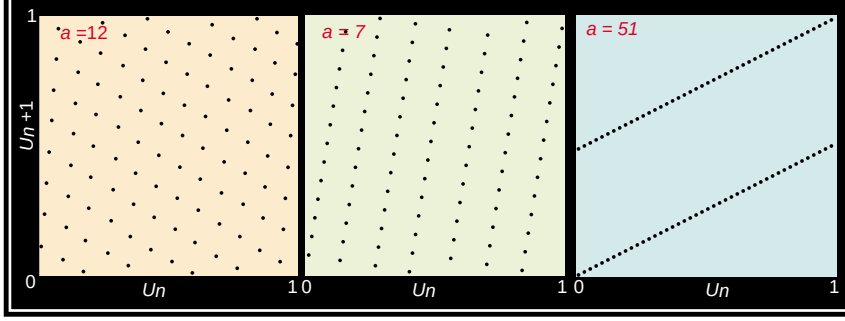
($z \mod M$ désigne le reste de la division de z par M)

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).

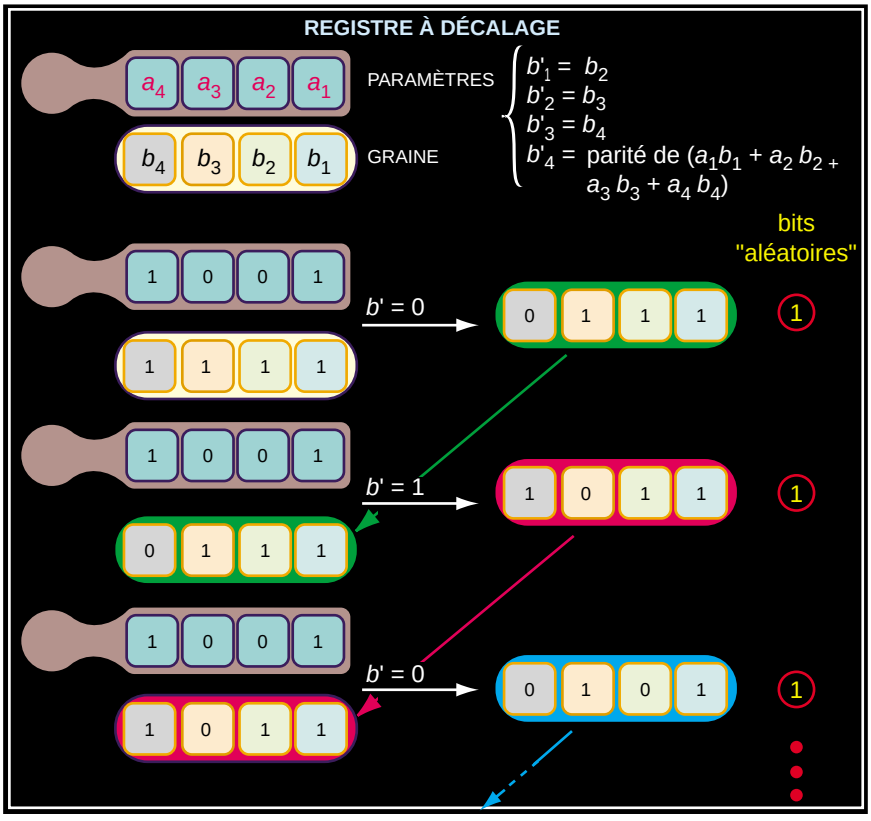
On peut transformer les entiers *x(i)* pour donner des nombres réels. En prenant par exemple $u(i) = x(i)/M$, on obtient une suite de nombres réels *u(i)* compris entre 0 et 1, dont la distribution est, en première approximation, uniforme : chaque sous-intervalle de [*y*, *z*] de [0, 1] contient *x(i)*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

3. LES DIFFÉRENTS HASARDS UTILISÉS

Hasard faible

Bon mélange, uniformité, longue période, satisfaction des tests statistiques. Peut être produit par des algorithmes rapides. Utile en simulation.

Hasard moyen

Imprévisibilité pour un observateur ne disposant que de moyens de calcul réalistes.
Peut être produit (vraisemblablement) par algorithmes moyennement rapides. Utile en cryptographie.

Hasard fort

Imprévisibilité totale, incompressibilité, contenu maximum en information.
Ne peut jamais être produit par algorithme,
mais vraisemblablement par des moyens physiques.
Utile en cryptographie et en théorie de l'information.

divise M divise aussi $a - 1$, et si M n'est pas divisible par 4, alors la période du générateur est maximale, c'est-à-dire M. Si M est un multiple de 4, il faut que $a - 1$ le soit aussi pour que la période soit maximale

Autrement dit, si vous vous y prenez bien, en partant d'une valeur $x(0)$ quelconque (entre 0 et $M - 1$), vous n'y reviendrez qu'après être passé par tous les nombres entre 0 et $M - 1$. Cette promenade par tous les nombres possibles entre 0 et $M - 1$ n'assure pas que la suite est vraiment désordonnée, mais montre qu'on ne va pas osciller entre des valeurs trop proches, ce qui est déjà rassurant.

Il s'agit d'une propriété dite structurelle. Les spécialistes ont analysé d'autres propriétés structurelles identifiant des conditions mathématiques suffisantes pour qu'une suite engendrée par des générateurs congruentiels linéaires soit acceptable, c'est-à-dire produise des points assez équitablement distribués ou ne s'alignant pas trop quand on les représente dans des espaces de test.

Dans une simulation utilisant n nombres pseudo-aléatoires, il est recommandé d'utiliser un générateur de période supérieure à n^2. On connaît des combinaisons de générateurs congruentiels linéaires engendrant des suites de période de 2^{200} et même plus. Récemment un générateur de période $2^{19937}-1$ a été proposé. Un mot de prudence : il est moins risqué d'utiliser un générateur simple dont on connaît les propriétés qu'un générateur complexe pour lequel rien n'est démontré.

En plus des propriétés structurelles prouvées, on met en œuvre (sans cette fois chercher à faire de démonstrations mathématiques) des tests statistiques de toutes sortes portant sur les fréquences des différents chiffres et des suites de ces chiffres. Des jeux de tests ont ainsi été développés (en particulier, la série *Diehard* de George Marsaglia) et sont disponibles sur Internet pour qui souhaite tester la qualité d'une suite aléatoire : http://stat.fsu.edu/~geo/diehard.html

Reste que les spécialistes des générateurs aléatoires sont d'accord : même la démonstration de bonnes propriétés structurelles et le passage réussi à travers une volumineuse batterie de tests statistiques ne garantissent pas qu'un générateur donné sera satisfaisant pour une simulation donnée. Mathématiques et tests peuvent disqualifier une suite, mais pas en assurer la qualité.

PRÉVISIBILITÉ

Cette situation est particulièrement dommageable en cryptographie, où l'on a souvent besoin de suites aléatoires et où le fait qu'une suite ne le soit pas vulnérabilise un procédé de codage. On raconte que, juste après la guerre, un algorithme de codage qui devait utiliser des clefs aléatoires toutes différentes fut utilisé deux fois de suite par les Russes avec la même clef, ce qui permit le déchiffrage des messages par les Américains et les conduisit à arrêter Klaus Fuchs, Julius et Ethel Rosenberg, Donald MacLean. L'officier russe responsable fut fusillé.

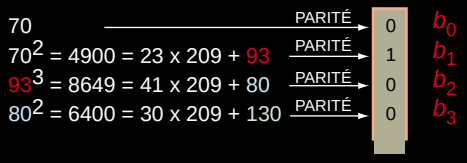
Les générateurs congruentiels linéaires sont depuis longtemps déconseillés en cryptographie. La raison tient à une série de résultats mathématiques qui indiquent qu'ils sont *prévisibles* : à partir de quelques points successifs de la suite $x(n)$, il est possible de calculer en un temps court les paramètres du générateur, et donc de prévoir son comportement futur avec une exactitude absolue, ce qui évidemment est contraire aux nécessités de la cryptographie. En cryptographie, conformément à l'idée que le hasard est imprévisible, il faut être certain que la suite aléatoire ne pourra pas être prédite *rapidement* à partir de la connaissance de quelques-uns de ses points.

«Rapidement», en informatique, signifie «ce qui peut être calculé en un temps plus petit qu'un polynôme de la taille des paramètres». Si la durée du calcul est inférieure à $n^3 + 2n^2 + 12$, où n est la taille des paramètres, on considère que le calcul est rapide. Si, en revanche, on ne sait faire un calcul qu'en un temps 2^n ou 3^n, etc., on considère que la longueur du calcul est inacceptable.

En utilisant cette idée, on arrive à une définition précise de la notion de «géné-

4. LE PROCÉDÉ DE L. BLUM, M. BLUM ET M. SHUB (BBS)

Le générateur BBS est vraisemblablement satisfaisant en cryptographie. Soit n un produit de deux entiers premiers, chacun de la forme $4m + 3$. On prend comme graine un entier x sans facteur commun avec n et l'on calcule $x(0) = x^2 \bmod n$. On définit ensuite $x(i + 1) = x(i)^2 \bmod n$. La parité de $x(i)$ donne la suite pseudo-aléatoire de bits proposée par BBS.



Si la graine est choisie aléatoirement, et si l'hypothèse QRA est vraie (elle exprime que «trouver les racines carrées modulo n est un problème difficile») alors la suite de bits $b_0 b_1 \ldots b_m$ proposée par BBS est imprévisible (aucun algorithme rapide ne fait mieux que le hasard pour prédire le m-ième digit à partir des précédents). Il en résulte, d'après un résultat de A. Yao de 1982, que la suite de bits $b_0 b_1 \ldots b_m$ est indiscernable, par des tests fonctionnant en temps polynomial, d'une suite produite par une variable aléatoire équitable.

rateur aléatoire cryptographique sûr». Est sûr un générateur dont la période est trop grande pour être explorée exhaustivement, et tel qu'en plus les moyens exigés pour mener une analyse conduisant à en prévoir le comportement sont inacceptablement coûteux.

Il faut remarquer qu'en cryptographie on accepte parfois qu'un générateur ne soit pas uniforme (par exemple, qu'il donne 45 pour cent de '0' contre 55 pour cent de '1') s'il est imprévisible, alors qu'en simulation la situation est inverse : l'uniformité est plus importante que l'imprévisibilité.

Existe-t-il des générateurs cryptographiquement sûrs? Vraisemblablement oui, et le générateur BBS décrit sur la figure 4 est un bon candidat comme générateur cryptographiquement sûr. Pourquoi vraisemblablement? Parce que les mathématiciens ont seulement réussi à établir que le BBS est sûr en admettant qu'une certaine conjecture est vraie (il s'agit de la conjecture sur la difficulté du calcul de la *résiduosité quadratique*). La conjecture est jugée très vraisemblable, et donc on est presque certain que BBS est un bon générateur aléatoire... mais pas absolument.

Cette situation où un résultat, conjecturé et jugé vraisemblable par les spécialistes, reste non démontré est fréquente en théorie de la complexité. Bien d'autres conjectures attendent le génie mathématique qui saura les élucider. La situation est inconfortable : les cryptographes utilisent des méthodes vraisemblablement bonnes, sans avoir d'assurances fermes à leur sujet. Cette remarque s'applique en particulier aux systèmes utilisés pour la carte à puce ou les transactions entre banques.

Qui a dit qu'on faisait trop de mathématiques? La création de suites aléatoires sûres est un exemple parmi d'autres (qu'on trouverait en physique) où apparaît l'insuffisance du développement des mathématiques contemporaines face aux besoins de l'industrie. Même sur des sujets économiquement cruciaux, car liés au commerce électronique et à la sécurité bancaire, on attend des avancées mathématiques.

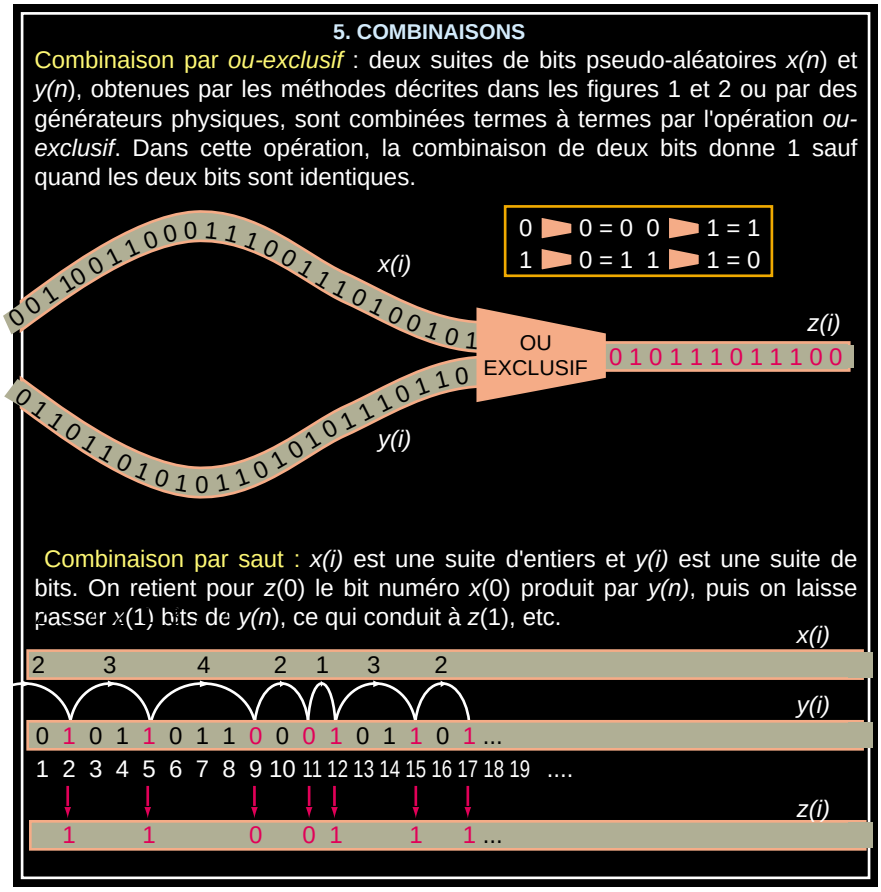
LE HASARD FORT

Ce hasard cryptographique (défini par la difficulté de la prédiction), sans doute atteint pas BBS, est le *hasard moyen*. Le *hasard fort* est défini par l'impossibilité absolue de la prédiction, quelle que soit la puissance de calcul utilisée. On a démontré, il y a quelques années, que cette impossibilité absolue est équivalente à l'impossibilité de la compression des données : une suite de nombres est aléatoire au sens fort s'il est impossible de la décrire en utilisant moins de place que celle qu'elle occupe.

Le malheur de ce hasard fort défini en 1965 par le mathématicien suédois Pier Martin-Löf est que, par nature, aucun programme déterministe d'ordinateur ne peut l'engendrer. Une procédure d'ordinateur (par nature déterministe) n'engendrera jamais de suites aléatoires au sens fort.

Dès 1951, avant même qu'elle n'ait pris un sens mathématique précis, von Neumann, conscient de cette difficulté fondamentale, écrivait : «Quiconque envisage une méthode arithmétique pour produire des chiffres aléatoires est en état de péché.»

À côté de l'impossibilité pour un programme déterministe d'engendrer une suite aléatoire au sens fort, on démontre que, si l'on réussit à disposer d'un phénomène physique non déterministe équiréparti (produisant des 0 et des 1 en proportion égale), alors en faisant travailler ce générateur on tombera presque certainement sur une suite aléatoire au sens de P. Martin-Löf (donc au sens le plus fort). Ce résultat semble être un jeu de mots – l'indéterminisme produit le hasard –, mais mathématiquement il lui correspond un énoncé précis qui n'est ni évident ni une boutade.



RÉALISATION DE TABLES DE NOMBRES ALÉATOIRES

Toutes ces raisons expliquent pourquoi on s'est souvent tourné vers des mécanismes physiques de production du hasard.

Il n'y a pas si longtemps, on éditait des tables de nombres aléatoires. En 1927, une table de 40 000 chiffres aléatoires fut publiée par L. Tippett. En 1939, un mécanisme physique fut utilisé par M. Kendal et B. Babington-Smith pour produire une table de 100 000 chiffres aléatoires. En 1955, la *RAND Corporation* publia une table d'un million de chiffres aléatoires engendrés, là encore, par une machine. Les tests statistiques utilisés pour mettre cette table à l'épreuve furent repris par J. Guilloud et M. Bouyer quand, en 1973, ils calculèrent un million de décimales de π. Le nombre π passa les tests. Cela ne signifie par que π est aléatoire au sens fort : le cryptographe saura reconnaître la suite des décimales de π et calculer les suivantes pour déchiffrer le message.

Les chiffres de π, contrairement à une légende tenace, ne conviennent donc ni comme hasard faible pour la simulation (trop compliqués à calculer), ni comme hasard moyen en cryptographie, où ils ne possèdent pas les propriétés d'imprévisibilité souhaitées.

Aujourd'hui la tradition de l'édition des tables de nombres aléatoires s'est adaptée, et un CD-ROM de plus de quatre milliards de bits aléatoires a été récemment édité par Marsaglia selon une méthode que je vais décrire. Des sites Internet proposent aussi des monceaux de bits aléatoires à qui veut bien.

Partant du constat que *(a)* les méthodes purement physiques produisent souvent des séquences avec des déviations de fréquences et autres défauts de mélange et qu'il faut donc s'en méfier en simulation, mais que *(b)* les méthodes par générateurs algorithmiques ne sont pas sûres, principalement à cause de leur prévisibilité, Marsaglia a additionné les deux procédés.

Il a pris des séries de bits engendrées par des procédés physiques et des séries générées par des générateurs algorithmiques ayant passé les tests de bon mélange, et a opéré un *ou-exclusif* entre les deux séries : '0' et '0' donnent '0' ; '0' et '1' donnent '1' ; '1' et '0' donnent '0' ; '1' et '1' donnent '1'. Cela combine les bonnes propriétés des deux méthodes : l'imprévisibilité est garantie par la composante physique, le bon mélange par la composante algorithmique. Bien sûr, ces séries ont été soumises aux batteries de tests dont on dispose. Notons que ces même batteries indiquaient clairement que la plupart des générateurs physiques ne fonctionnaient pas de manière très satisfaisante.

Même s'il a été produit avec soin et qu'il sera certainement utile, le CD-ROM n'est malheureusement pas la solution ultime. D'une part, pour la simulation, il se peut qu'on ait besoin de séries aléatoires encore plus longues que celles du CD-ROM. D'autre part, en cryptographie, le fait que le CD-ROM soit disponible à tous est quelque peu inquiétant !

Les générateurs physiques fondés sur le bruit thermique tel QNG de la Société *ComScire* (http://shell.rmi.net/~comscire), ou sur la diode Zener exploitant l'effet tunnel des électrons, comme ORION (http://195.11.242.249/ave/com/orion/home.html) ont encore de beaux jours devant eux. Selon les applications, on conseillera donc de les utiliser en les combinant (en utilisant un *ou-exclusif*, par exemple) avec des méthodes simples (par exemple, des générateurs congruentiels linéaires) ou des méthodes dont l'imprévisibilité est admise.

L'ERREUR DE NETSCAPE

Dans les applications où peu de bits aléatoires sont nécessaires, l'idée ancienne de mesurer un paramètre dépendant du contexte physique est bonne, surtout si on la combine avec une autre.

C'est ce qui est fait dans les machines à sous des casinos où un générateur simple tourne en permanence, dont on ne prend en compte le résultat qu'à l'instant précis où le joueur joue (ce qui assure une forme d'imprévisibilité). Des mécanismes de mesure de la date ou des caractéristiques de la frappe de celui qui est devant l'ordinateur peuvent être utilisés pour obtenir un petit nombre de bits aléatoires, et c'est ce que fait PGP (*Pretty Good Privacy*, le fameux logiciel de cryptographie utilisé sur *Internet*) pour engendrer ses clefs aléatoires.

Mais là encore attention : il y a deux ans, le protocole de communication confidentielle de *Netscape*, célèbre logiciel de navigation sur *Internet*, a été cassé par des étudiants. Ils avaient remarqué que les clefs étaient engendrées par lecture de certaines parties de la mémoire de l'ordinateur selon un procédé qui ne pouvait conduire qu'à des clefs appartenant à une partie assez réduite de l'ensemble des clefs possibles (ensemble *a priori* trop grand pour être exploré exhaustivement). Les clefs possibles étant en nombre réduit, l'approche exhaustive devenait utilisable et permettait de décrypter les messages secrets. L'erreur a été corrigée.

Cette mésaventure s'est produite fréquemment, et c'est pourquoi une note intitulée *Randomness Recommendation for Security*, rédigée par des experts en sécurité, insiste sur une erreur commune : en piochant dans une très grande base de données (par exemple, le disque dur de l'ordinateur de l'utilisateur), on ne tombe pas toujours sur des bits aléatoires.

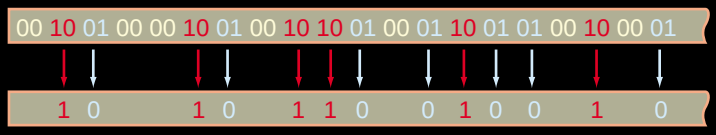
Les disques des ordinateurs (surtout s'ils sont neufs) comportent de grandes zones vides ; de plus, les fréquences d'utilisation des caractères ne sont pas uniformes, ce qui conduit à fausser gravement l'uniformité *a priori* sur les clefs engendrées et rend parfois, comme dans le cas évoqué ci-dessus, envisageable une approche exhaustive.

Les recommandations des experts sont là encore de mélanger plusieurs sources non corrélées, en utilisant le *ou-exclusif* déjà mentionné ou d'autres fonctions plus perfectionnées. Ils insistent aussi sur la taille des clefs qui protègent contre les attaques par énumération exhaustive : elles doivent comporter au moins 50 bits (10^9 clefs pourraient bientôt être traitées par seconde, et donc plus de 10^{15} par mois, ce qui correspond à 50 bits).

Ces exemples confirment que, dans le domaine de la génération de suites aléatoires, courtes ou longues, à usage cryptographique ou pas, les pièges sont partout présents et infiniment variés : on ne se méfie jamais assez du hasard.

6. UNIFORMISATION D'UN GÉNÉRATEUR BIAISÉ

Lorsqu'on utilise un générateur physique, il est souvent difficile d'avoir autant de «0» que de «1» (c'est-à-dire un générateur équilibré). Il y a plus de 40 ans, von Neumann a proposé un moyen très simple de corriger un tel défaut : prendre les bits deux par deux ; supprimer tous les couples «00» et tous les couples «11» ; transformer tous les couples «01» en «0» et tous les couples «10» en «1». Exemple :



On perd quelques bits, mais on corrige un éventuel déséquilibre entre le nombre moyen de «0» et le nombre moyen de «1».

Si la suite initiale donne «1» avec une probabilité p et «0» avec une probabilité $1 - p$, le couple «01» a pour probabilité $p(1–p)$ comme le couple «10», et donc la suite obtenue par le procédé de von Neumann donne bien «0» et «1» avec la même probabilité, c'est-à-dire 1/2.

Jean-Paul DELAHAYE est directeur adjoint du Laboratoire d'informatique fondamentale de Lille du CNRS.

e-mail : delahaye@lifl.fr

L. BLUM, M. BLUM et M. SHUB, *A simple unpredictable pseudo-random number generator,* SIAM J Computing 15.2 pp. 364-383, 1986.

J.-P. DELAHAYE, *Le fascinant nombre Pi*, Éditions *Pour La Science*, Paris, 1997.

L. GOUBIN et J. PATARIN, *La génération d'aléas sur l'ordinateur,* in *Quadrature*, oct.-nov.-déc. 1997, pp. 27-36.

P. L'ÉCUYER, *Random number generation. Hanbook on Simulation*, Ed. Jerry Banks, Wiley, 1997.

B. SCHNEIER, *Cryptographie appliquée*, Thompson Publishing, 1995.

C. P. SCHNORR, *A survey of the theory of random sequences*, in *Basic Problems in Methodology and Linguistics*, Dordrecht, pp. 193-210, 1977.