REGARDS

LOGIQUE & CALCUL

Le défi des faibles complexités

Comme les très petites durées ou longueurs, les faibles complexités sont délicates à estimer. Paradoxalement, leur évaluation exige des calculs colossaux.

Jean-Paul DELAHAYE

Prenons un objet défini par un grand nombre de données, par exemple une photo numérique ou le texte d'un livre. Le bon sens nous souffle que plus grande est la quantité d'information nécessaire pour décrire l'objet, plus il est « complexe ». Le corollaire est que plus votre objet est complexe, plus sa transmission prendra du temps.

Si l'image à laquelle nous nous intéressons est toute noire (par exemple un carré d'un million de pixels noirs), la brève description « l'image a pour dimension 1 000 1 000 et tous ses pixels sont noirs » suffit pour la connaître avec une parfaite précision. Dans le cas d'une photographie de visage, même les meilleures méthodes de description exigeront des milliers de bits d'information. Le visage est plus complexe et la valeur de cette complexité déterminera le coût de la transmission.

Les théoriciens ont précisé cette façon naturelle et intuitive d'envisager la complexité d'une suite de 0 et de 1. C'est le concept de complexité introduit vers 1965 en même temps par Andreï Kolmogorov et Gregory Chaitin. La complexité de Kolmogorov d'un objet numérique s est la taille $K(s)$ du plus court programme informatique, une description optimale de s, qui, exécuté, reconstitue exactement s. La taille d'un fichier compressé de l'objet s est une mesure de sa complexité. En effet, l'algorithme de décompression, partant de ce fichier, reconstitue toutes les données de l'objet s (on n'envisage ici que la compression sans perte) ; nous considérons donc ce fichier compressé comme un programme dont la taille est approximativement celle du programme le plus court donnant s.

Étudiée depuis la décennie 1960, la complexité de Kolmogorov a été appliquée en physique, en biologie et même en économie. Elle permet de concevoir clairement ce qu'est le contenu en informations d'un objet.

Échecs pour les faibles complexités

Avec la complexité de Kolmogorov, nous savons caractériser mathématiquement une suite aléatoire, ce dont était incapable la théorie classique des probabilités : une suite aléatoire s de n bits d'information (une suite de n fois 0 ou 1) est une suite dont la complexité de Kolmogorov $K(s)$ est environ n. Une suite infinie aléatoire est une suite dont aucun début (la suite des n premiers bits) n'est sensiblement compressible. Au contraire, la suite infinie 01010101... n'est pas aléatoire, car on peut définir de façon concise ses $2n$ premiers bits par « n fois 01 ».

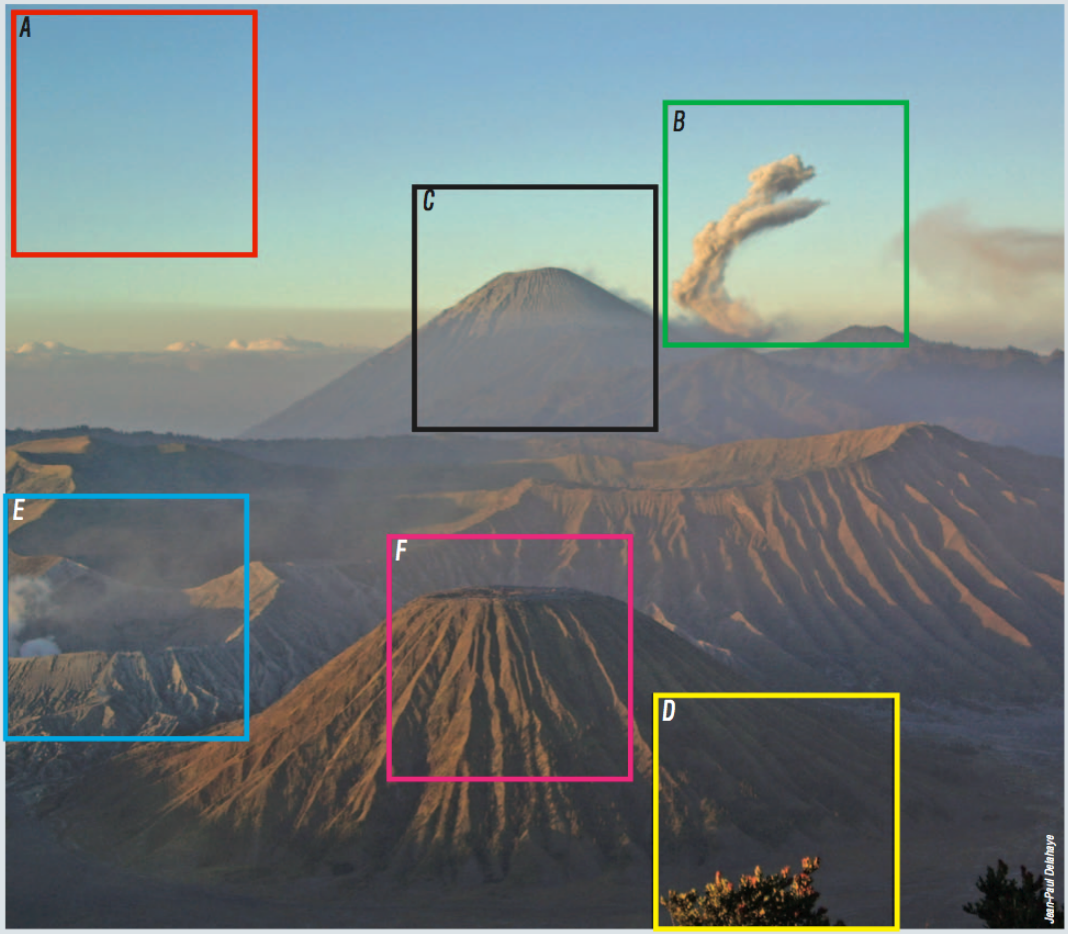
Le calcul des valeurs approchées de la complexité de Kolmogorov, grâce aux algorithmes de compression de données sans perte, rend le concept utilisable : la figure 1 propose des exemples de calcul de la complexité de Kolmogorov pour des images numériques. Certaines méthodes de classification de textes, de musiques et de séquences génétiques ont été déduites d'algorithmes de compression de données.

Cette théorie a cependant un défaut : elle ne permet de mesurer la complexité d'un objet s que s'il est assez grand (supérieur à quelques centaines de bits). La raison de cette limitation est que la définition de $K(s)$ dépend du langage de programmation utilisé pour obtenir la « longueur du plus court programme » : selon le langage choisi, la complexité de Kolmogorov subit des variations qui ont peu d'importance lorsqu'il s'agit de longues séquences de 0 et de 1, mais qui enlèvent tout sens aux résultats concernant les petites séquences.

Pourtant, il semble censé d'envisager la complexité de Kolmogorov des séquences courtes. En effet, il paraît clair que, par exemple, la séquence de sept bits 1001101 est plus complexe que la séquence 0000000, et que la séquence 0001000 est d'une complexité intermédiaire à celles des deux précédentes. Nous avons donc une idée intuitive d'un classement par complexité croissante des séquences courtes. Comment faire pour que la théorie donne un sens à ce que notre intuition perçoit, et que cette théorie s'applique à la fois aux séquences courtes et longues ?

Le problème à résoudre est un problème de thermomètre : parmi tous les instruments de mesure qui conduisent à des évaluations de la complexité de Kolmogorov, mais qui diffèrent par des constantes additives, n'y en aurait-il pas un meilleur, ou devant être préféré, au moins sur un plan

1. Comprimer pour mesurer



La complexité de Kolmogorov d'un fichier informatique *s* est la taille du plus court programme, mesurée en nombre de bits, capable de produire le fichier *s*.

Un algorithme de compression de données transforme un fichier informatique *s* en un fichier « compressé » *s'*. Certains algorithmes sont dits « sans perte », car lorsqu'on décompresse le fichier *s'*, celui-ci redonne exactement le fichier initial *s*. Ces algorithmes sans perte nous intéressent, car le fichier compressé *s'* peut être assimilé à un programme minimal. La taille du fichier compressé *s'* est donc une valeur approchée de la complexité de Kolmogorov du fichier initial *s*.

Voici une photo du volcan *Bromo* situé à l'Est de l'île de Java en Indonésie. Plusieurs parties de l'image de même taille géométrique (mesurée en nombre de pixels) ont été sélectionnées et confiées à un algorithme de compression de données sans perte, l'algorithme PNG.

La taille des fichiers compressés varie en fonction de la complexité de la partie de photo sélectionnée. Le ciel presque uniforme *(image A)* a une complexité de 324 kilo-octets (1 kilo-octet = 8192 bits). Le panache de projections émis par le volcan *(image B)* est un peu plus complexe (400 kilo-octets).

Les autres parties sont encore plus complexes : l'image *C* a une complexité de 416 kilo-octets ; *D*, de 420 kilo-octets ; *E*, de 512 kilo-octets ; et *F*, de 544 kilo-octets.

2. Complexités

Certaines méthodes séduisantes utilisées pour le calcul de la complexité des courtes séquences sont insuffisantes. En effet, elles ne coïncident pas avec la complexité de Kolmogorov quand la longueur des séquences augmente, ce qui est nécessaire pour que la mesure soit satisfaisante.

L'entropie de Shannon est définie par l'expression $-\sum p_i \log p_i$ (où p_i est une fréquence). Avec une suite binaire, cela donne $[-p_1 \log p_1 - p_2 \log p_2]$, où p_1 et p_2 sont respectivement les densités de 0 et de 1 dans la séquence. Avec cette mesure, la suite 01010101010101010101 est la plus complexe possible pour une suite longueur 20 (elle a autant de 0 que de 1). Or la suite 10010111010100001011, qui a la même entropie de Shannon et la même longueur, est plus complexe. L'entropie de Shannon ne prend en compte que les fréquences de 0 et de 1 ; ce n'est pas suffisant pour savoir si une séquence est simple ou complexe. C'est une mesure statistique qui ne prend même pas en compte les répétitions.

La complexité par facteurs d'une suite *s* est déterminée par le nombre de facteurs différents qu'on trouve dans *s*. Un facteur est une suite de symboles contigus. Ainsi, la séquence 00000 possède cinq facteurs différents : 0, 00, 000, 0000, 00000. Elle a comme complexité 5. La séquence 01011 possède les 11 facteurs différents 0, 1, 01, 10, 11, 010, 101, 011, 0101, 1011, 01011 et elle est donc plus complexe.

Cette mesure est mauvaise, car elle attribue à certaines suites régulières une grande complexité. Ainsi, la suite de Champernowne obtenue en écrivant les entiers en binaire les uns derrière les autres (0 1 10 11 100 101 110 111 1000 etc.) met en défaut la mesure par facteurs. Par exemple, la suite 01101110010111011111000 (les entiers jusqu'à 8) n'est pas complexe, car on peut la définir brièvement et pourtant, elle contient un très grand nombre de facteurs. La mesure de complexité par facteurs ne peut pas être adoptée : une mesure générale doit être équivalente à la complexité de Kolmogorov lorsque les longueurs tendent vers l'infini, or ce n'est pas du tout le cas ici.

pratique ? L'instabilité des thermomètres les plus simples (ceux fondés sur la longueur minimale des programmes) les rend inutilisables pour le classement des faibles complexités.

La solution est apparue il y a environ cinq ans. Un thermomètre nouveau, mis au point au Laboratoire d'informatique fondamentale de Lille selon les idées et les programmes de Hector Zenil, donne des réponses satisfaisantes pour les courtes et les longues séquences. Il définit une complexité conforme à l'idée générale de Kolmogorov pour les grandes complexités et classe, comme on l'attend, 0000000, 0001000, 1001101, ou toutes les séquences pour lesquelles notre intuition n'hésite pas.

Le nouveau thermomètre que nous détaillerons possède un défaut : il est extrêmement coûteux en calcul. En pratique, il ne donne de résultats que pour les séquences très courtes et, de ce point de vue, les méthodes par compression restent incontournables. Comme en astronomie où, selon le type d'objets et leur éloignement, on utilise un procédé ou un autre pour évaluer les distances, dans les mesures de complexité, on doit adopter des méthodes hybrides avec recollement de plusieurs procédés.

Un critère fondé sur la probabilité de production

L'idée sous-jacente aux thermomètres pour les basses complexités est fondée sur une propriété remarquable de la complexité de Kolmogorov : plus un objet est simple, plus il est produit fréquemment quand on fait fonctionner un ordinateur en lui donnant des programmes choisis au hasard.

Un langage de programmation universel étant donné (*Java, C, Pascal, Basic* sont de tels langages), nous imaginerons que nous tirons au hasard des programmes écrits dans ce langage (en fait on peut écrire les programmes en binaire et tirer un programme au hasard revient à tirer une suite de 0 et de 1 jusqu'à avoir un programme complet). Nous tomberons souvent sur des programmes grammaticalement incorrects (donc ne fonctionnant pas) ou sur des programmes qui tourneront sans produire aucun résultat. Ces programmes ne nous intéressent pas : seuls ceux qui produisent en sortie une suite finie de 0 et de 1 et s'arrêtent sont pris en compte. Les expériences montrent qu'en procédant ainsi, on obtient plus souvent 0000000 que 1001101 : la probabilité d'obtenir en sortie une suite de grande complexité est plus faible que la probabilité d'obtenir une suite de faible complexité. Cette probabilité se nomme mesure de Solomonoff-Levin, en l'honneur de Ray Solomonoff (1926-2009) et Leonid Levin (professeur à l'Université de Boston), qui ont identifié ce phénomène.

Un théorème, clef des thermomètres pour les faibles complexités, fait le lien avec la complexité de Kolmogorov. Ce théorème affirme que $K(s)$ est environ égal à $-\log_2 SL(s)$, où $\log_2$ est le logarithme en base 2 (on a $\log_2(x) = \log x/\log 2$). En termes mathématiques, $K(s)$ étant une complexité de Kolmogorov définie par un langage de programmation et $SL(s)$ la mesure de Solomonoff-Levin associée au même langage, il existe une constante c, indépendante de la séquence s, telle que :

$$|-\log_2 SL(s) - K(s)| < c.$$

À l'infini, $-\log_2 SL(s)$ et $K(s)$ ne diffèrent au plus que d'une constante fixée une fois pour toutes. La quantité $-\log_2 SL(s)$ est donc aussi intéressante qu'une complexité de Kolmogorov mesurée par les plus courts programmes et coïncide presque avec elle pour les longues séquences.

Le premier avantage à considérer le nombre $-\log_2 SL(s)$ plutôt que $K(s)$ provient de ce que c'est un nombre bien plus stable. En changeant de langage de programmation, ou seulement en modifiant légèrement la définition d'un langage de programmation choisi, nous ferons beaucoup varier $K(s)$ pour les petites séquences, ce qui empêche, nous l'avons vu, de donner un sens à ce qu'on obtient. En revanche, le nombre $-\log_2 SL(s)$ ne varie que très peu. La raison est que ce nombre résulte d'un lissage opéré sur un très grand nombre de données dû au fonctionnement d'une multitude de programmes.

Cependant, l'argument le plus puissant qui fait s'intéresser à la mesure de Solo-

monoff-Levin – $\log_2 SL(s)$ est qu'elle donne ce que nous attendons du classement des séquences courtes : ainsi, les trois séquences 0000000, 0001000 et 1001101 sont classées dans cet ordre. Pour les petites séquences, cette mesure probabiliste est stable et conforme à notre idée de la complexité ; pour les grandes, elle est, d'après le théorème mentionné, conforme à la meilleure mesure de complexité unanimement admise, la complexité de Kolmogorov. Que demander de plus ?

Le calcul de $SL(s)$ s'obtient en faisant fonctionner un très grand nombre de programmes, qui seront soit tirés au hasard, soit pris dans un ensemble de programmes énumérés systématiquement (ce qui revient au même). En cumulant les résultats, on obtient expérimentalement un calcul approché de la distribution $SL(s)$, donc des nombres $-\log_2 SL(s)$ donnant la complexité des petites séquences. Le procédé est le même que celui qu'on appliquerait pour évaluer si une loterie est truquée ou non : on lancerait la loterie un très grand nombre de fois, et, à partir des résultats obtenus, on observerait, par exemple, que le 13 sort plus souvent que les autres numéros. À l'aide d'un grand nombre de tirages, on évaluerait même la probabilité de sortie de chaque numéro et bien sûr, plus le nombre de tirages expérimentaux serait grand, meilleure serait la précision obtenue pour les probabilités de chaque numéro.

On comprend cependant les difficultés de la méthode. Pour évaluer expérimentalement les probabilités d'une loterie à 20 numéros, il faut faire des milliers de tirages. Pour évaluer le nombre $SL(s)$ pour les 128 séquences binaires de sept chiffres ou les 1024 séquences binaires de longueur dix, il faut faire un nombre de tirages encore plus massif, et la limite de ce qu'on peut envisager pratiquement est rapidement atteinte, quelle que soit la puissance informatique dont on dispose.

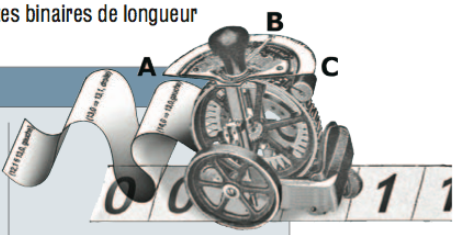
Les machines de Turing à la rescousse

Comment H. Zenil a-t-il procédé ? Il a utilisé une machine à calculer aussi primitive que possible, mais suffisamment puissante pour que tout programme puisse y être exécuté. Ces « machines de Turing », introduites par Alan Turing en 1936, jouent depuis un rôle fondamental en informatique théorique et en logique mathématique, car on n'a pas su faire plus simple et plus général pour étudier la notion d'algorithme. Ces machines constituent un langage de programmation, parce que décrire une machine de Turing est équivalent à écrire un programme informatique. Passer par ces machines est intéressant pour le calcul de $SL(s)$, car elles permettent une programmation d'une redoutable efficacité : des programmes courts mènent des calculs non triviaux *(voir la description d'une machine de Turing sur la figure 3)*.

Le nombre d'états d'une machine de Turing détermine son pouvoir de calcul. Les machines à un état ne peuvent faire que des calculs simples, par exemple inverser les 0 en 1 et les 1 en 0 d'une séquence qu'on leur soumet.

Les machines à deux états sont déjà plus intéressantes. Leur nombre est 10 000 (les formules compliquées de dénombrements de ces machines donnent ce résultat et c'est un hasard qu'il soit un chiffre aussi rond), ce qui bien sûr ne peut donner accès qu'à une vue extrêmement sommaire de $SL(s)$. En faisant fonctionner toutes les machines de Turing à trois états, les choses deviennent plus sérieuses. Ces 7 529 526 machines produisent une approximation de $SL(s)$ qui n'est pas absurde et qui montre qu'on est sur la bonne voie. Par exemple, parmi les suites binaires de longueur

3. La machine de Turing



Une machine de Turing est un mécanisme de calcul qui peut se trouver dans une série d'états internes (ici trois états notés A, B, C). La machine possède une tête de lecture-écriture qui se déplace et modifie une à une les cases d'un ruban infini (une sorte de mémoire externe). En fonction de l'état interne et de ce que la tête de lecture-écriture observe sur le ruban, la machine décide (en suivant des règles constituant le programme de la machine et fixées une fois pour toutes) de changer ce qui est écrit dans la case sous sa tête de lecture-écriture et de se déplacer vers la droite ou vers la gauche d'une unité en passant dans un autre état interne. Voici une machine à trois états A, B, C possédant un programme de quatre instructions :

(A, 0 → B, 1, droite)
(B, 0 → C, 1, gauche)
(C, 1 → B, 1, droite)
(B, 1 → arrêt)

L'instruction (A, 0 → B, 1, droite) signifie que lorsque la machine est dans l'état A et qu'elle lit un 0, alors elle passe dans l'état B en remplaçant le 0 par un 1 et se déplace d'une case vers la droite. Lorsque l'état dans lequel la machine passe est l'état *arrêt*, elle cesse toute action. La machine donnée en exemple opère un calcul élémentaire : placée sur un ruban infini couvert de 0, elle mène quelques opérations d'écriture et de déplacements et s'arrête en laissant ...001100... Cette machine produit la séquence $s = 11$. Voyons le détail des calculs.

A
.....0 0 0 0 0 0 0 0 ...

La machine, dans l'état A, lit un 0. L'instruction (A, 0 → B, 1, droite) lui indique qu'elle doit changer le 0 en 1, passer dans l'état interne B et se déplacer vers la droite :

B
... 0 0 0 1 0 0 0 0 ...

L'instruction (B, 0 → C, 1, gauche) lui fait écrire un second 1 et la conduit dans l'état C :

C
... 0 0 0 1 1 0 0 0 ...

L'instruction (C, 1 → B, 1, droite) donne maintenant :

B
... 0 0 0 1 1 0 0 0 ...

L'instruction (B, 1 → arrêt) fait cesser toute activité.

La machine a produit la séquence $s = 11$ sur les cases qu'elle a modifiées. Dans l'évaluation, que l'on fait avec toutes les machines de Turing à trois états, des complexités des suites à deux éléments, cette machine augmentera la probabilité de la séquence binaire 11.

4. Tableau de résultats

Hector Zenil a fait fonctionner les 11019960576 machines de Turing à quatre états et en a déduit, par un énorme calcul sur ordinateur, la fréquence des séquences finies produites. Il a obtenu des valeurs proches de la mesure de Solomonoff-Levin $SL(s)$ reliées à la complexité de Kolmogorov par la formule $K(s) = -\log_2 SL(s)$. Puis H. Zenil a classé par complexité croissante les valeurs approchées de $K(s)$ des séquences courtes.

0	2,285794	1111	8,644805	00000	11,791904	10110	12,755680
1	2,285794	0001	9,019632	11111	11,791904	01010	12,830138
00	3,287643	0111	9,019632	00001	12,513528	10101	12,830138
01	3,287643	1000	9,019632	01111	12,513528	00110	12,946395
10	3,287643	1110	9,019632	10000	12,513528	01100	12,946395
11	3,287643	0101	9,029529	11110	12,513528	10011	12,946395
000	5,735457	1010	9,029529	00010	12,554899	11001	12,946395
111	5,735457	0010	9,040613	01000	12,554899	00101	12,979816
001	5,791928	0100	9,040613	10111	12,554899	01011	12,979816
011	5,791928	1011	9,040613	11101	12,554899	10100	12,979816
100	5,791928	1101	9,040613	00100	12,694473	11010	12,979816
110	5,791928	0110	9,263820	11011	12,694473	00011	13,177551
010	5,867773	1001	9,263820	01001	12,755680	00111	13,177551
101	5,867773	0011	9,276355	01101	12,755680	11000	13,177551
0000	8,644805	1100	9,276355	10010	12,755680	11100	13,177551

six, le calcul de $SL(s)$ mené en faisant fonctionner toutes les machines de Turing à trois états et en examinant ce qu'elles produisent indique que les deux séquences les plus probables (donc les moins complexes) sont à égalité 000000 et 111111, ce qu'on attendait. Viennent ensuite dans l'ordre les séquences suivantes :

– 000001 à égalité avec 100000, 111110 et 011111 ;

– 000100 à égalité avec 001000, 111011 et 110111 ;

– 001001 à égalité avec 100100, 110110 et 011011 ;

– 010110 à égalité avec 101001, 100101 et 011010.

Un tel classement est assez subtil, mais naturel : placer un 1 derrière cinq 0 est un peu plus simple que de placer un 1 avec trois 0 devant et deux derrière. Si vous en doutez, comparez la longueur des deux descriptions que je viens de formuler :

– « un 1 derrière cinq 0 »

– « un 1 avec trois 0 devant et deux derrière ».

Cependant, le classement obtenu avec les machines de Turing à trois états ne donne de résultats que pour un petit nombre de séquences, celles qui sont produites par au moins une machine. Il y a 128 séquences différentes ainsi produites. Cela laisse toutes les autres sans probabilité, ce qui revient à considérer que ces autres séquences sont infiniment complexes.

Pour obtenir plus de précision et surtout pour qu'un plus grand nombre de séquences soient classées, il faut sauter un pas et faire fonctionner toutes les 11 019 960 576 machines de Turing à quatre états.

Pour chacune, on doit déterminer ce qu'elle fait en simulant son fonctionnement. Il y a deux possibilités : soit la machine produit une séquence de 0 et de 1 et s'arrête, et alors elle contribuera au calcul de l'approximation de $SL(s)$ (comme un tirage de loterie) ; soit elle ne s'arrête jamais et la machine ne sert à rien. Malgré l'utilisation de plusieurs astuces pour écourter le calcul, les programmes de H. Zenil ont dû fonctionner neuf jours durant, à raison de plusieurs milliers de machines par seconde, pour obtenir cette nouvelle version approchée de $SL(s)$.

Calcul colossal et castors affairés

Le fait que de nombreuses machines calculent indéfiniment sans jamais rien produire a pu être maîtrisé grâce à la connaissance qu'on a du *Castor affairé* à quatre états. On dénomme Castor affairé à n états, la machine de Turing à n états qui calcule le plus longtemps avant de s'arrêter. Le Castor affairé à trois états calcule durant 21 étapes et donc, toute machine ayant trois états calcule moins de 22 étapes, ou alors calcule indéfiniment. Le Castor affairé à quatre états calcule durant 107 étapes. Le Castor affairé à cinq états n'a pas encore été identifié avec certitude, mais on pense qu'il calcule durant 47 176 870 étapes.

Comme le Castor affairé à quatre états calcule 107 étapes, toute machine de Turing à quatre états qui ne s'est pas arrêtée au bout de 107 étapes ne s'arrêtera jamais. Lors du calcul de $SL(s)$, on peut interrompre le fonctionnement d'une telle machine, puisqu'il est certain qu'elle ne produira aucune séquence. Cette prise en compte est évidemment essentielle pour ne pas perdre de temps en faisant fonctionner inutilement des machines qui ne contribuent pas au résultat recherché.

Les résultats obtenus avec les machines à quatre états concernent cette fois une volumineuse famille de séquences binaires. Précisément 1 824 séquences différentes sont produites par les machines de Turing à quatre états et ce sont donc 1 824 séquences dont on a su évaluer la complexité par utilisation de l'approximation ainsi calculée de $SL(s)$ et de la formule $-\log_2 SL(s)$ qui détermine leur complexité de Kolmogorov approchée. Notons que cette complexité de Kolmogorov pour séquences courtes est un nombre réel pas forcément entier. C'est en fait un avantage, puisque cela permet d'avoir un classement plus fin avec moins d'*ex aequo*.

Voici quelques observations sur les résultats obtenus :
– Pour les séquences prises en exemple au début de l'article, tout se passe bien. La séquence 0000000 est plus probable que 0001000, qui elle-même est plus probable que 1001101. Les complexités déduites des probabilités pour ces trois séquences sont conformes à ce que l'intuition exigeait.
– Chaque suite ayant une longueur comprise entre 1 et 8 est produite au moins une fois et le calcul approché avec les machines de Turing à quatre états lui attribue une probabilité non nulle, donc une complexité finie.
– Pour les 128 séquences déjà classées par les machines à trois états, le classement obtenu avec les machines à quatre états reste le même, sauf dans 17 cas. En général, les déplacements opérés entre les deux classements conduisent à des positions plus conformes à l'intuition. En considérant des machines à quatre états, on améliore ce qu'on avait obtenu avec les machines à trois états : c'est une validation de la méthode.
– Les séquences produites les plus longues ont pour longueur 16. En voici une : 1101010101010101. Bien sûr, ce sont des suites assez peu complexes, car les suites de longueur 16 les plus complexes ne sont pas produites par une machine de Turing à quatre états.
– En général, plus une séquence est longue, plus sa complexité est grande, mais ce n'est pas une règle absolue : par exemple, la séquence 0000000000 (dix fois le 0) est évaluée moins complexe que la séquence de neuf bits 000011100.
– De petites imperfections persistent. En passant de trois à quatre états, on a notablement progressé, mais pour avoir un classement parfait incluant par exemple toutes les séquences de longueur allant jusqu'à dix, il faudrait maintenant mener un calcul complet avec les machines à cinq états, à la limite de ce qu'on peut envisager aujourd'hui. On espère l'entreprendre prochainement. Bien sûr, il ne correspondra qu'à une marche de plus dans une quête infinie.

Pour disposer d'une méthode pratique qui attribue une complexité à toute séquence finie, il faut accepter d'associer plusieurs idées. Trois idées semblent suffire :

(a) Pour les séquences courtes (jusqu'à huit ou dix bits, peut-être un peu plus), la méthode des machines de Turing ;

(b) Pour les séquences de longueur moyenne (de dix bits à quelques centaines de bits) un découpage en tranches ramenant au cas précédent ;

(c) Pour les séquences plus longues, une méthode à base d'algorithmes de compression.

Recollement délicat

Le travail est délicat et on tombera sur des cas où l'évaluation sera peu satisfaisante. Mais le principe général semble acquis et tout n'est plus qu'une question de perfectionnement et d'efforts : prises en compte de machines de Turing ayant plus d'états ; utilisation simultanée d'une multitude d'algorithmes de compression, etc. Ces efforts n'auront jamais de fin, mais nous disposons maintenant de moyens pour le calcul de la complexité de Kolmogorov des séquences binaires de toutes les longueurs, moyens qu'on adaptera aux séquences plus générales de chiffres décimaux, de lettres, de nombres entiers, etc.

Aujourd'hui, une grande variété de méthodes spécifiques, fondées sur des principes particuliers et bien souvent *ad hoc* sont utilisées pour calculer et comparer des complexités d'objets discrets *(voir deux exemples sur la figure 4)*. C'est le cas en informatique dans le domaine des algorithmes pour suites pseudo-aléatoires, en traitement d'images, en génétique, en linguistique, en psychologie, etc. Les avancées du programme de recherche décrit ici mettront à disposition de tous une méthode générale de calcul de la complexité de Kolmogorov qui, en plus d'être homogène (car l'unité de mesure est toujours le bit d'information), sera fondée sur ce qui est reconnu comme la théorie universelle de l'information. Cette théorie longtemps considérée comme inutilisable (du fait de l'existence de nombreux énoncés indécidables concernant $K(s)$) se révèle au fur et à mesure des années praticable et utile. Il est certain pourtant qu'on est loin d'en avoir tiré toutes les applications possibles. ■

L'AUTEUR

Jean-Paul DELAHAYE est professeur à l'Université de Lille et chercheur au Laboratoire d'informatique fondamentale de Lille (LIFL).

✔ BIBLIOGRAPHIE

H. Zenil, Pages Internet donnant accès aux articles cités ci-dessous : http://www.mathrix.org/zenil/

H. Zenil et J.-P. Delahaye, An algorithmic information theoretic approach to the behaviour of financial markets, themed issue on « Nonlinearity, Complexity and Randomness », *Journal of Economic Surveys*, 2011.

H. Zenil, Compression-based investigation of the dynamical properties of cellular automata and other systems, *Journal of Complex Systems*, vol. 19(1), 2010 : http://arxiv.org/abs/0910.4042

J.-P. Delahaye et H. Zenil, A glance into the structure of algorithmic complexity : Numerical evaluation of the complexity of short strings by small Turing machines, 2011 : http://arxiv.org/pdf/1101.4795

J.-P. Delahaye et H. Zenil, On the algorithmic nature of the world, dans G. Dodig-Crnkovic et M. Burgin (éds.), *Information and Computation*, World Scientific, 2010.

J.-P. Delahaye et H. Zenil, On the Kolmogorov-Chaitin complexity for short sequences, *Randomness and Complexity : From Leibniz to Chaitin* (C.S. Calude, éd.), World Scientific, pp. 123-130, 2007.

J.-P. Delahaye, Classer musiques, images, textes et génomes, *Pour la Science*, pp.90-95, mars 2004.