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 :

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

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.

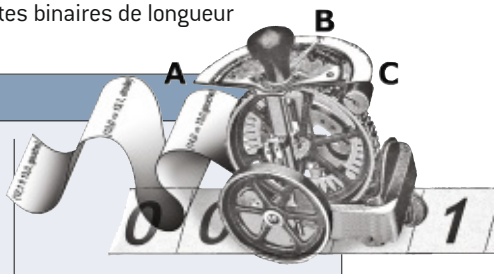
Les machines de Turing à la rescousse

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.