Les surprises de la complexité algorithmique

Bernard Chazelle

Department of Computer Science, Princeton University

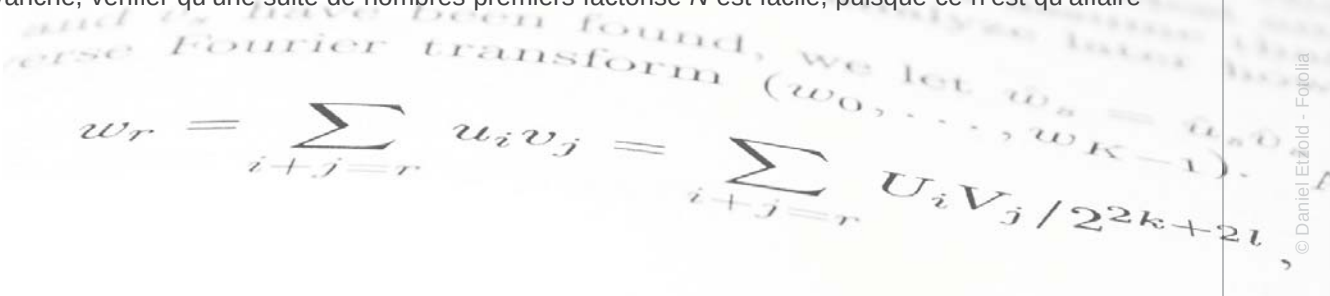
© DR

Les deux dernières décennies ont vu l'émergence d'une théorie de l'aléa algorithmique dont les conséquences sur la notion de preuve ont une portée philosophique à même de surprendre. Voici, en quelques mots, ce dont il s'agit.

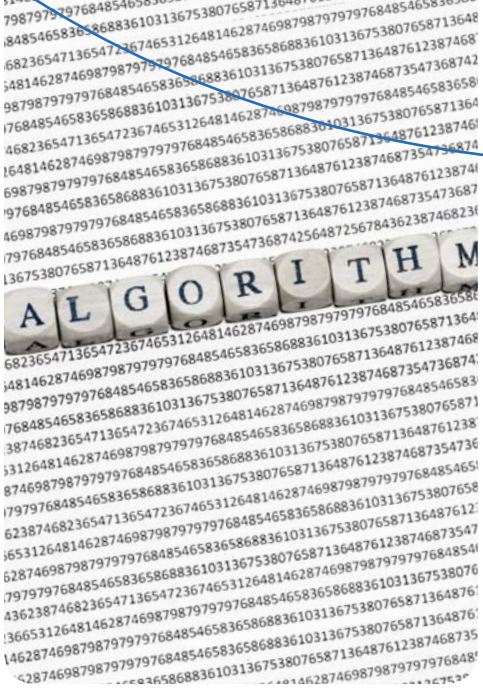
P, NP et tout cela

En informatique théorique, la complexité d'un problème désigne la quantité de ressources nécessaires à sa solution. Il est d'usage de se limiter à trois types de ressources : le temps, l'espace et l'aléa. L'algorithme scolaire pour la multiplication, par exemple, requiert un temps quadratique : en effet, multiplier deux entiers de n chiffres nécessite un nombre d'opérations élémentaires de l'ordre de n^2. L'espace requis est également quadratique, mais il est facile de le rendre proportionnel à n. Existe-t-il un algorithme plus rapide ? Curieusement, en faisant un détour par la transformée de Fourier, on peut réduire le temps d'exécution à une fonction proche de n log n. Cet algorithme est certes trop compliqué pour détrôner celui enseigné à l'école primaire, mais sa rapidité justifie sa présence au cœur de certains codes cryptographiques.

Un problème est dit polynomial s'il peut être *résolu* en un nombre d'étapes proportionnel à n^c, où n dénote la taille du problème et c est une constante. C'est notamment le cas de la multiplication, de la transformée de Fourier, de la programmation linéaire et de bien d'autres tâches d'usage courant. Ces problèmes constituent la classe P, dont la contrepartie, la classe NP, regroupe les problèmes dont la solution peut être *vérifiée* en temps polynomial. Considérons le problème de factoriser un entier N de n chiffres. Il y a lieu de croire (et d'espérer) que c'est un problème difficile, c'est-à-dire non résoluble en temps polynomial ; en revanche, vérifier qu'une suite de nombres premiers factorise N est facile, puisque ce n'est qu'affaire

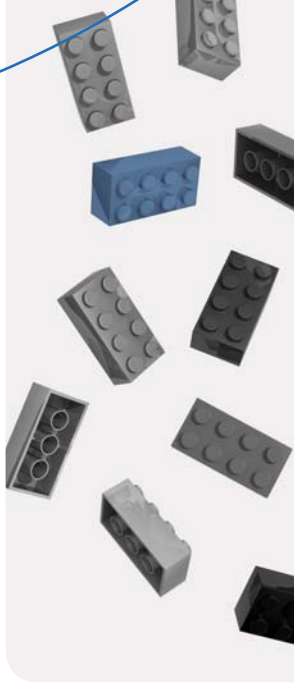


© Daniel Etzold - Fotolia



© Denis Junker - Fotolia

© Dreaming Andy - Fotolia



de multiplications : la factorisation est donc dans NP. Mais pourquoi espérer que ce soit une opération difficile ? Parce que le sort de toute la cryptographie à clé publique, et donc du commerce électronique, en dépend. En rendant la factorisation facile (autrement dit, polynomiale), les ordinateurs quantiques promettent de bouleverser cet ordre. Encore faut-il qu'ils voient le jour...

Le paradoxe de l'aléa

Le choix du concept polynomial dans la définition de P provient de deux observations. D'une part, pour des raisons pratiques, seuls les algorithmes polynomiaux sont viables pour les ordinateurs. D'autre part, la classe P est invariante par l'opération de composition : il est ainsi possible d'assembler des algorithmes pour en créer d'autres - un peu comme on assemble des Lego® - afin de réduire un problème à un autre. À cette propriété se retrouve associé le concept de NP-complétude, qui désigne les problèmes les plus difficiles dans NP - une classe extrêmement riche : automatiser la preuve des théorèmes mathématiques, par exemple, est NP-complet, de même que résoudre un système d'équations quadratiques modulo 2, simuler le repliement des protéines ou, encore, colorier un graphe en assignant des couleurs à ses sommets de façon à rendre toutes les arêtes bichromatiques. La classe des problèmes NP-complets jouit d'une propriété remarquable : savoir résoudre n'importe lequel d'entre eux en temps polynomial permettrait d'en faire de même pour tous les autres. Cette universalité est à même de surprendre. Elle implique qu'une solution rapide pour le coloriage de graphe, autrement dit P = NP, entraînerait automatiquement une méthode rapide pour découvrir une démonstration de l'hypothèse de Riemann. Il n'est donc guère surprenant qu'un consensus se soit formé autour de la conjecture P ≠ NP. Cette inégalité, sans aucun doute l'un des plus grands défis scientifiques de notre époque, formalise l'intuition que trouver est plus dur que vérifier, que la créativité ne peut être automatisée, que composer la *Messe en si mineur* est plus ardu que d'apprécier le génie de cette musique.

© Prosetisen - Fotolia

Hypothèse A : il existe des problèmes naturels que l'on ne sait pas, et ne saura jamais, résoudre en temps polynomial.

Et l'aléa là-dedans ? Il existe un nombre de problèmes importants pour lesquels l'usage de la randomisation semble incontournable. Par exemple, armés de *bits* aléatoires, on peut facilement approximer le volume d'un polyèdre en haute dimension ou trouver un nombre premier de *n bits*. Dans tous ces cas, c'est l'usage de *bits* aléatoires qui accélère le temps de calcul d'exponentiel à polynomial. Retirez l'aléa, et les problèmes redeviennent difficiles. Bien qu'on ne sache l'établir, la conclusion suivante s'impose :

Hypothèse B : il existe des problèmes faciles à résoudre avec des algorithmes probabilistes mais qui, sans recours à la randomisation, redeviennent difficiles.

Le terme « facile » évoque ici la distinction entre un temps polynomial et un temps exponentiel : que l'algorithme soit compliqué ou pas n'est pas pertinent à l'hypothèse, même si, en pratique, cet élément n'est pas négligeable. Par exemple, « *décider si un entier de* n *chiffres est premier* » appartient à la classe P, puisqu'il existe un algorithme déterministe pour répondre à cette question en temps polynomial *n* ; mais l'algorithme n'étant pas particulièrement simple, on se tourne en pratique vers un autre algorithme, probabiliste, qui est, lui, d'une simplicité enfantine. Les *bits* aléatoires sont donc *indispensables* en pratique, mais sont-ils *nécessaires* pour réduire la complexité de certains problèmes d'exponentiel à polynomial, comme le prévoit l'hypothèse B ? Il y a une abondance d'indices suggérant qu'en effet ils le sont.

Maintenant convaincu que les deux hypothèses A et B ne peuvent être que vraies, le lecteur se voit confronté à un sérieux dilemme : A et B sont, en fait, incompatibles. Il existe une opinion relativement consensuelle que seule A est valide. Pourquoi ? L'intuition est la suivante : l'existence de problèmes difficiles permet de manufacturer des *bits* de manière déterministe qu'aucun algorithme polynomial ne peut distinguer de *bits* produits de façon parfaitement aléatoire. Autrement dit, on ne peut pas faire la différence entre de l'aléa parfait et du pseudo-aléa, c'est-à-dire de l'information d'apparence aléatoire. En utilisant ces *bits* déterministes, on peut alors « dérandomiser » tout algorithme probabiliste sans perte de temps : l'aléa devient alors un luxe, utile, mais accessoire. Soulignons au passage que la qualité du quasi-aléa nécessaire à ce subterfuge est nettement supérieure à ce que l'on rencontre en statistique : par exemple, les décimales de π forment de bien piètres générateurs de quasi-aléa. De fait, la production déterministe de *bits* d'apparence aléatoire est une affaire complexe, qui repose sur une théorie riche de plusieurs décennies de recherche active.

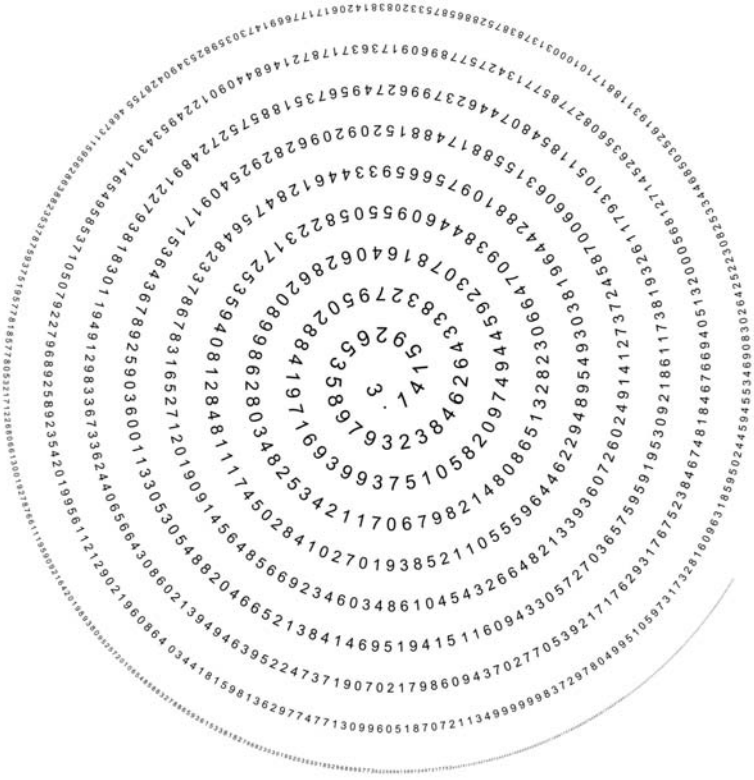
Vérifier une preuve est trivial

S'il est donc probable que la randomisation ne soit pas pertinente à la séparation de P et de NP, il est un autre domaine où l'aléa, en revanche, joue un rôle déterminant : celui des preuves. Tout mathématicien comprend bien ce que constitue une preuve d'un théorème. La même notion s'étend à des questions

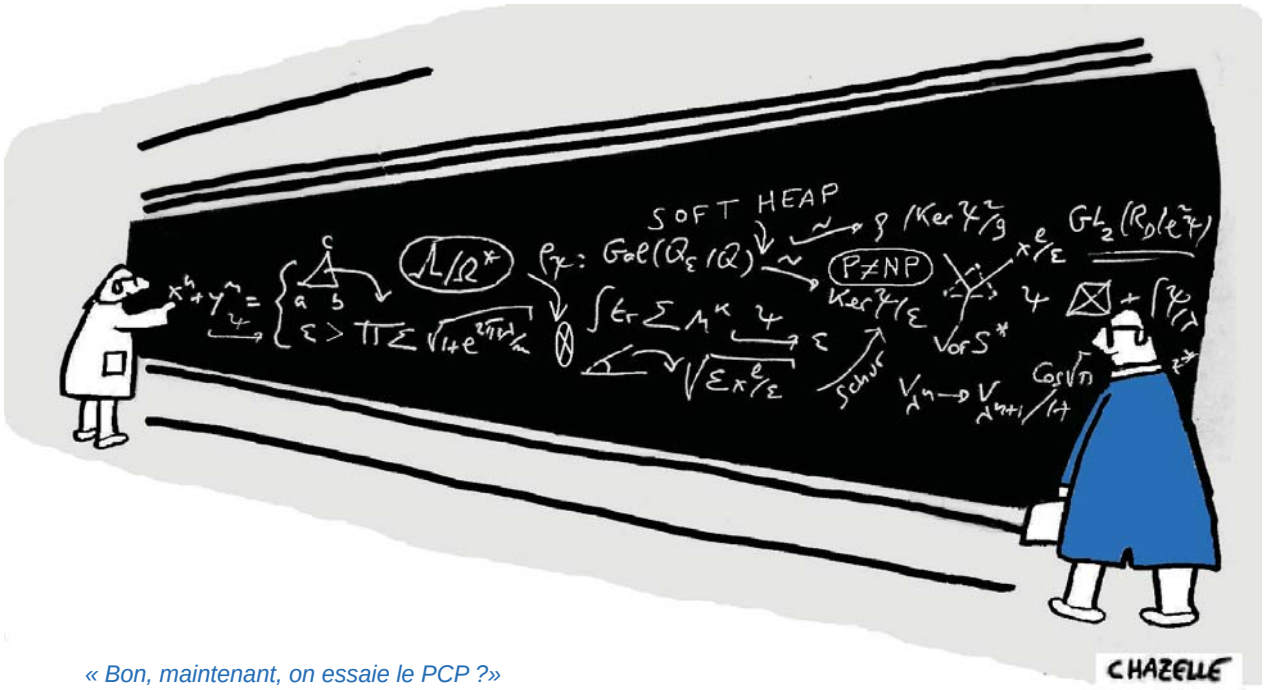
propres à la cryptographie. Il est facile d'imaginer des situations où l'on désire convaincre un tiers que nous sommes en possession d'un code secret, sans pour autant le révéler explicitement ; une preuve à divulgation nulle se charge de cette tâche. Dans le même esprit, je peux vous convaincre que je sais colorier un graphe avec trois couleurs sans pour autant vous révéler la moindre information sur le coloriage en question. Moyennant l'existence de « fonctions à sens unique » - l'outil fondamental de la cryptographie -, qui veulent que l'antécédent d'une image est difficile à trouver, tout problème dans NP admet une preuve à divulgation nulle.

Une notion encore plus étonnante est celle de « preuve à vérification instantanée » (*Probabilistically Checkable Proof*, PCP). Supposons que j'aie calculé la millionième décimale de π, qui se trouve être 1 : comment vous convaincre en quelques secondes que c'est en effet 1 ? Je peux vous donner accès à un ordinateur qui contient le programme de calcul de π, vous tapez 1 000 000 et, après un temps court de calcul, la réponse s'inscrit sur l'écran : oui, c'est bien 1. C'est rapide, mais cela ne constitue pas une preuve, car vous êtes obligé de me faire confiance sur le bon fonctionnement du programme, de l'ordinateur, du compilateur, etc. À l'inverse, je peux écrire dans un (gros) livre toutes les étapes de calcul du développement en série de π qui m'a amené à cette millionième décimale : elles constituent une preuve de mon affirmation qui ne requiert aucune confiance de votre part, mais il vous sera impossible de vérifier rapidement que je n'ai fait aucune erreur de calcul !

Il n'y aurait donc pas d'alternative : ou vous me faites confiance, et la vérification se passe très vite, ou vous vous méfiez, et votre tâche devient alors longue et rude. En fait, cette dichotomie est illusoire : je peux en effet vous convaincre très rapidement de la véracité de mon calcul sans exiger de vous la moindre confiance, grâce à l'algorithme PCP. Son utilisation me permet de réécrire la longue preuve dans un format spécial dont vous pourrez extraire au hasard une dizaine de caractères : leur examen rapide vous conduira à convenir que j'ai raison, ou à rejeter mon affirmation. Le miracle est que, dans tous les cas de figure, vous n'errerez qu'avec une probabilité infime, par exemple de l'ordre d'une chance sur mille. Bien sûr, plus vous piochez de caractères, plus cette probabilité d'erreur chute, avec une rapidité exponentielle. Ce résultat est général : si je prétends avoir une démonstration de l'hypothèse de Riemann, je peux également vous en convaincre de façon quasi instantanée. Existe-t-il une version non probabiliste de



© senoldo - Fotolia



« Bon, maintenant, on essaie le PCP ?»

l'algorithme PCP ? Non, l'aléa y est indispensable. Sans accès à des *bits* aléatoires, vous êtes condamné à lire ma preuve dans sa totalité.

Ce qui est étonnant dans ce résultat, c'est que ma preuve au format PCP est forcément immune à un nombre considérable d'erreurs : en effet, si vous ne piochez que 10 caractères au hasard, je peux me permettre des milliers d'étapes erronées sans avoir crainte d'être pris en flagrant délit. Encore plus surprenant, si ma preuve est entièrement correcte mais prouve autre chose, par exemple la conjecture de Poincaré, vous vous en apercevrez tout de suite. Formellement, le théorème PCP s'exprime sous la forme NP = PCP (log *n*, 1). Traduction : « *Toute solution d'un problème dans NP (une preuve) peut être écrite en temps polynomial dans sa taille* n *de telle façon qu'elle puisse être vérifiée en consultant un nombre constant de ses* bits*. Le nombre de* bits *aléatoires est au plus logarithmique en* n. »

Mathématiquement, l'algorithme PCP étend le concept de code correcteur d'erreurs aux fonctions : un code traditionnel permet de maintenir l'intégrité syntaxique d'un signal en présence d'erreurs, tandis qu'un code PCP maintient l'intégrité sémantique d'une preuve sujette à erreur. La vérification instantanée, quant à elle, repose sur la notion de codes « localement décodables ». Ce résultat profond a des ramifications épistémologiques évidentes : une tradition philosophique remontant à Platon définit la connaissance comme une croyance vraie justifiée. La justification est nécessaire pour éviter la connaissance par hasard. L'algorithme PCP nous apprend que l'aléa rend la vérification de la justification triviale. Au-delà même des mathématiques et de l'informatique, cela semble être une contribution philosophique majeure !