L'EFFICACITÉ TROMPEUSE DES ALGORITHMES GALACTIQUES

Une méthode de calcul peut être la meilleure en théorie, mais totalement inutile pour toute application dans le monde réel.

L'AUTEUR

JEAN-PAUL DELAHAYE
professeur émérite à l'université de Lille et chercheur au laboratoire Cristal (Centre de recherche en informatique, signal et automatique de Lille)

Classer les diverses façons d'aller vers l'infini est un problème délicat qui conduit parfois les mathématiciens à de dangereuses simplifications. Cela produit des situations paradoxales quand il s'agit d'évaluer l'efficacité des algorithmes, c'est-à-dire l'efficacité des programmes informatiques.

Si deux fonctions $f(n)$ et $g(n)$ vont vers l'infini quand n tend vers l'infini, il est naturel de dire que $f(n)$ y va plus vite que $g(n)$ si la limite de $f(n)/g(n)$ tend vers l'infini. On dit alors que $f(n)$ domine $g(n)$.

Par exemple, la fonction $f(n)=2^n$ va plus vite vers l'infini que $g(n)=n^2$ et la différence se voit en calculant quelques valeurs :

$f(10)=1024 > g(10)=100$
$\rightarrow f(10)/g(10)=10{,}24$
$f(20)=1048576 > g(20)=400$
$\rightarrow f(20)/g(20)=2621{,}44$
$f(100)=1{,}25\times10^{30} > g(100)=10000$
$\rightarrow f(100)/g(100)=1{,}25\times10^{26}$.

On montre facilement qu'un polynôme de degré p dont le premier coefficient est positif (par exemple $2n^5+3n$ de degré 5) va plus vite vers l'infini qu'un polynôme de degré q dont le premier coefficient est positif (par exemple $5n^4+12$, de degré 4) quand $p>q$. On montre aussi que n domine $\log(n)$, qui domine $\log(\log(n))$, et que 4^n domine 3^n, qui domine 2^n, etc.

Deux polynômes de même degré dont les premiers coefficients sont positifs vont tous les deux vers l'infini, mais aucun ne domine l'autre. Exemple : si on compare $2n^2$ et n^2+n, aucun ne domine l'autre car $[2n^2]/[n^2+n]=2/[1+1/n]$ tend vers 2 quand n tend vers l'infini. Plus généralement, si C est un nombre positif, alors entre $f(n)$ et $g(n)=C\times f(n)$ aucun ne domine l'autre ; multiplier par une constante n'a pas d'importance à l'infini !

MESURER L'EFFICACITÉ DES ALGORITHMES

En informatique, il faut évaluer la quantité de calculs nécessaires pour résoudre un problème. On s'intéresse donc à la façon dont la fonction qui indique cette quantité de calculs va vers l'infini quand la quantité de données utilisées par un programme augmente. C'est le domaine de la complexité des algorithmes et des programmes.

Considérons un cas d'importance réelle : on veut classer n nombres donnés dans une liste en désordre, par exemple (6, 23, 11, 7). On peut utiliser la méthode de l'insertion : on prend les nombres un par un que l'on retire de la liste des données, et on construit une nouvelle liste classée en insérant chaque élément nouveau au bon endroit dans la liste qui se construit. Avec notre exemple cela donne (6), puis (6, 23), puis (6, 11, 23), puis (6, 7, 11, 23). Pour classer n nombres de cette façon, le nombre d'étapes est, dans ce cas, mesuré par le nombre des comparaisons entre deux nombres entiers, soit au plus $f(n)=n^2/2-n/2$ (*voir l'encadré 1 pour les détails*). Cela semble raisonnable, pourtant une méthode un peu plus compliquée, dénommée « tri par séparation-fusion » (*merge-sort*, en anglais), fait le travail

Jean-Paul Delahaye a récemment publié : **Au-delà du Bitcoin** (Dunod, 2022).

1 LA COMPLEXITÉ DES ALGORITHMES

Souvent, on connaît plusieurs algorithmes pour résoudre un même problème. De manière à choisir celui qui calculera le plus rapidement les solutions attendues, on évalue leur complexité en nombre de calculs. Ces études aboutissent par exemple à des affirmations comme « le nombre d'opérations à faire pour résoudre un problème de taille n augmente comme n », ou « comme n^2 », ou « comme 2^n », etc. Ces affirmations oublient parfois de mentionner des constantes qu'il faudrait placer devant n ou n^2, ou 2^n, alors qu'un algorithme qui calcule en un nombre d'étapes égal à $2 \times n$ est en pratique très différent d'un algorithme qui calcule en $10^{20} \times n$ étapes. Les constantes ont de l'importance ! Les constantes associées à certains algorithmes sont même si grandes dans certains cas que jamais on ne pourra les utiliser ; ce sont les algorithmes galactiques.

© Golden Dayz/Shutterstock

en au plus $g(n) = n\lceil \log_2(n) \rceil$ comparaisons entre entiers. La notation $\lceil x \rceil$ désigne le plus petit entier supérieur ou égal à x. Ainsi, $\lceil 2{,}41 \rceil = 3$ et $\lceil 5 \rceil = 5$. La complexité du tri par séparation-fusion est meilleure que celle de la méthode de l'insertion, car la fonction $f(n)$ domine la fonction $g(n)$. On en conclut qu'il faut choisir la seconde méthode.

Au moment de programmer un algorithme pour mettre en ordre une liste, il est vraiment très important de comprendre cette différence. En effet : en utilisant la méthode de l'insertion vous ne pourrez pas classer une liste de un million de nombres, alors qu'avec le tri séparation-fusion, ce sera assez facile. Utiliser la mauvaise méthode pour des données de taille modérée retarde seulement l'obtention du résultat, mais pour des données volumineuses, cela empêche d'arriver au résultat !

Compter le nombre de comparaisons entre deux entiers pour mesurer la complexité des algorithmes de tri n'est pas très précis, car comparer 4 et 7 est un calcul plus simple que comparer 520 327 et 520 329, et il serait bon de prendre en compte cette différence. Si on souhaite mesurer avec une meilleure exactitude la complexité d'un algorithme de classifications d'une liste d'entiers, il faut prendre en compte le nombre d'opérations élémentaires que fait la machine, et par exemple le nombre de comparaisons entre deux chiffres décimaux. Pour comparer 520 327 et 520 329 il faut d'abord comparer 5 et 5, puis 2 et 2, etc. C'est seulement quand on arrive à 7 et 9 qu'on peut conclure. Dans le cas général, pour comparer deux nombres entiers ayant k chiffres, on devra parfois faire k comparaisons élémentaires entre paires de chiffres.

La conséquence de cette analyse plus fine est que pour classer deux listes d'entiers qui peuvent chacun avoir jusqu'à k chiffres, le nombre de comparaisons entre chiffres dans les pires cas pour les deux méthodes envisagées n'est pas $f(n) = n^2/2 - n/2$ et $g(n) = n\lceil \log_2(n) \rceil$, mais $f_k(n) = k(n^2/2 - n/2)$ et $g_k(n) = k \times n\lceil \log_2(n) \rceil$. Il faut multiplier par la constante k. Or dans le cas de ce problème de tri et dans la plupart des évaluations de complexité, les constantes comme k sont négligées. Plutôt que de l'évaluer, on dira que les algorithmes pour classer des nombres ont « en ordre de grandeur » une complexité de $f(n) = n^2/2 - n/2$ avec la méthode d'insertion, et $g(n) = n\lceil \log_2(n) \rceil$ pour la méthode de séparation-fusion. Quand on néglige la constante, on parle souvent de « complexité asymptotique ». Cette façon d'ignorer les constantes est justifiée en théorie, puisque, comme on l'a vu plus haut, il semble que « multiplier par une constante n'a pas d'importance à l'infini ! »

UNE NÉGLIGENCE DANGEREUSE ?

Cependant ne pas prendre en compte les constantes est dangereux : en pratique, si la constante est très grande cela peut inverser le classement par efficacité des algorithmes pour certaines valeurs de n. Si, par exemple, on compare deux algorithmes A et B dont l'un a pour complexité $f_A(n) = 5 \times n^2$ et l'autre $g_B(n) = 1\,000\,000 \times n$, on simplifiera en disant que la complexité asymptotique de A est $f(n) = n^2$ et que celle de B est $g(n) = n$, ce qui amènera à la conclusion qu'il faut préférer B puisque $f(n) = n^2$ domine $g(n) = n$. Pourtant,

dans une telle situation, pour toute valeur $n<200000$, l'algorithme A doit être préféré à l'algorithme B, car : $n<200000 \Rightarrow 5\times n<1000000 \Rightarrow 5\times n^2<1000000\times n$.

On pourrait croire que ce genre de situations ne se produit pas, et donc qu'on a raison le plus souvent de ne prendre en compte que les complexités asymptotiques sans s'occuper des constantes à placer devant. C'est faux. Le type de situations envisagées avec A et B se produit parfois, et on connaît même des situations où il faut changer l'algorithme à préférer pour une valeur de n tellement grande qu'en pratique on n'aura jamais dans une application réelle à préférer celui que la complexité asymptotique désigne comme le meilleur. C'est à ce sujet que le terme « d'algorithme galactique » a été introduit par Ken Regan en 2010.

Un algorithme galactique est un algorithme dont la complexité asymptotique (c'est-à-dire où on néglige la constante) semble bonne mais dont le coût réel quand on prend en compte la constante est si élevé qu'il interdit son utilisation pratique pour tous les cas susceptibles de se présenter dans notre galaxie !

Les algorithmes galactiques les plus étonnants concernent la multiplication des nombres entiers. L'histoire de ces algorithmes est extraordinaire et constitue un exemple remarquable établissant qu'il faut se méfier des évidences intuitives qu'on ne démontre pas, et des simplifications presque toujours sans conséquences… qui en ont parfois.

RETOUR À L'ÉCOLE PRIMAIRE

L'algorithme de multiplication qu'on apprend à l'école pour multiplier deux nombres de n chiffres a une complexité asymptotique de n^2. On le voit quand on examine comment on pose la multiplication comme nous l'avons appris à l'école. On fait appel à nos tables de multiplication apprises par cœur, et on y fait appel exactement n^2 fois, puisqu'on prend chaque chiffre du premier facteur qu'on multiplie par chaque chiffre du second facteur pour effectuer le calcul. Il faut bien sûr ajouter quelques additions et reports de retenues, mais asymptotiquement la complexité est effectivement n^2 même quand on affine l'analyse des calculs. Pendant très longtemps, aucun des algorithmes connus

2 LA COMPLEXITÉ DES TRIS

Parmi les multiples méthodes permettant de classer des nombres donnés en désordre pour en faire une liste de nombres triés par taille croissante, il y a le *tri par insertion* et le *tri par séparation-fusion*.

Tri par insertion

On prend les éléments de la liste à trier un par un et on construit une liste agencée par ordre croissant en insérant chaque nouvel élément au bon endroit. Cette introduction dans une liste déjà triée de longueur n demande au plus n comparaisons entre deux nombres. Au total, le tri par cette méthode naturelle demande donc au plus $1 + 2 + \ldots + (n - 2) + (n - 1) = (n^2 - n)/2$ comparaisons entre deux nombres.

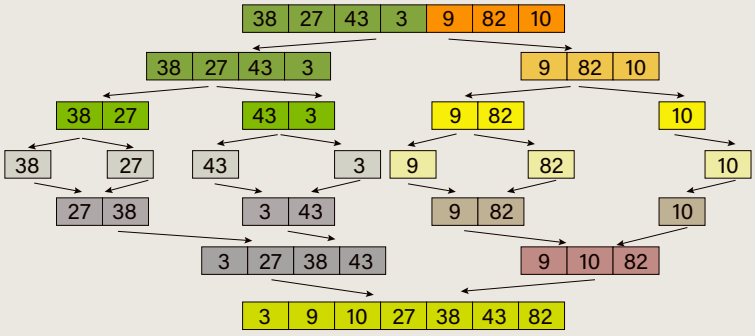
Exemple de liste à trier :
[13, 21, 5, 30, 2, 10]
13 est le premier élément de la liste
→ [13][21, 5, 30, 2, 10]
21 est inséré dans la liste
→ [13, 21][5, 30, 2, 10]
5 est inséré dans la liste
→ [5, 13, 21][30, 2, 10]
30 est inséré dans la liste
→ [5, 13, 21, 30][2, 10]
2 est inséré dans la liste
→ [2, 5, 13, 21, 30][10]
10 est inséré dans la liste
→ [2, 5, 10, 13, 21, 30][]
La liste est triée

Tri par séparation-fusion

Cette méthode fonctionne en deux phases. Dans une première phase la liste est séparée en deux sous-listes de longueurs égales ou presque égales et on recommence jusqu'à ce qu'il n'y ait plus que des sous-listes à un élément *(comme sur la figure ci-dessous)*. Dans la seconde phase, on fusionne petit à petit les listes de manière à les classer. L'opération de base dans cette phase est la fusion de deux listes triées l'une dans l'autre pour obtenir une seule liste triée. Cette opération de base exige au plus n comparaisons entre nombres où n est le nombre total d'éléments des deux listes qu'on insère l'une dans l'autre comme dans l'exemple ci-dessous :

Exemple de listes à fusionner
[1, 4, 7, 9] et [3, 5, 6, 10]
[] [1, 4, 7, 9] + [3, 5, 6, 10] et 1 < 3
→ [1] [4, 7, 9] + [3, 5, 6, 10] et 4 > 3
→ [1, 3] [4, 7, 9] + [5, 6, 10] et 4 < 5
→ [1, 3, 4] [7, 9] + [5, 6, 10] et 7 > 5
→ [1, 3, 4, 5] [7, 9] + [6, 10] et 7 > 6
→ [1, 3, 4, 5, 6] [7, 9] + [10] et 7 < 10
→ [1, 3, 4, 5, 6, 7] [9] + [10] et 9 < 10
→ [1, 3, 4, 5, 6, 7, 9] [] + [10]
→ [1, 3, 4, 5, 6, 7, 9, 10]

En bleu, la liste par fusion, qui se construit progressivement par l'introduction un à un d'éléments nouveaux, à chaque fois grâce à une comparaison entre deux éléments. Le nombre d'étapes de séparations pour passer d'une liste de n éléments à des sous-listes d'un élément est $\lceil \log_2 n \rceil$, où la notation $\lceil x \rceil$ désigne le plus petit entier supérieur ou égal à x. Au total, le nombre de comparaisons pour trier une liste de n nombres selon la méthode de séparation-fusion est donc au plus $g(n) = n\lceil \log_2 n \rceil$. Cette seconde méthode de tri doit donc être préférée à la première.



et utilisés pour effectuer des multiplications à la main ou avec un ordinateur n'a fait mieux en complexité asymptotique que ce n^2.

ARGUMENTS DE L'IGNORANCE

En 1956, le grand mathématicien russe Andreï Kolmogorov émit même l'hypothèse qu'il était impossible de faire mieux que n^2. Kolmogorov considérait que si, au sujet d'un problème aussi élémentaire et fondamental, une amélioration était possible, elle aurait certainement été découverte et utilisée depuis longtemps. L'argument qui pose que lorsqu'une affirmation n'a pas été démontrée comme fausse c'est qu'elle est vraie est parfois dénommé *argumentum ad ignorantiam*, ou l'« argument de l'ignorance ». En mathématiques plus qu'ailleurs, il faut s'en méfier : en 1960, Anatoli Karatsuba, un jeune étudiant russe de 23 ans, découvrit une astuce algébrique qui réduit la complexité asymptotique de la multiplication et la fait passer à $n^{\log_2(3)}$, soit environ $n^{1,595}$ ce qui est mieux que n^2 (*voir l'encadré 3*). Ici, pas de problème avec la constante, qu'on ne prend pas la peine de rechercher. Dès qu'on doit multiplier des nombres de quelques dizaines de chiffres, le nouvel algorithme est réellement utile et, par exemple, pour multiplier deux nombres de mille chiffres le temps de calcul est au moins 10 fois plus petit avec l'astuce de Karatsuba qu'avec une méthode en n^2.

Puisque, concernant la multiplication, certains progrès sont envisageables, d'autres apparurent. En 1971, Arnold Schönhage et Volker Strassen, deux mathématiciens allemands, proposèrent une méthode dont la complexité asymptotique est $n\log(n)\times\log(\log(n))$. Les deux mathématiciens ont alors formulé la conjecture que $n\times\log(n)$ est la complexité asymptotique la meilleure possible pour la multiplication d'entiers ayant n chiffres, conjecture qui reste non démontrée.

Indiquons que l'algorithme de Karatsuba est d'usage courant en particulier dans le domaine de la cryptographie. Celui de Schönhage et Strassen, pour certains calculs comme la recherche de nombres premiers très grands ou les calculs records de décimales du nombre π, se révèle meilleur que celui de Karatsuba et est donc utilisé… sur Terre.

TOUJOURS MIEUX

Se posait quand même la question de savoir si le résultat de 1971 n'était pas le meilleur. La réponse a attendu 2007, année où le mathématicien américain Martin Fürer a proposé une multiplication dont la complexité asymptotique bat celle de la méthode de Schönhage et Strassen. La complexité asymptotique de la méthode de Fürer est un peu compliquée à décrire, mais elle est intermédiaire entre $n\log(n)\times\log(\log(n))$ et $n\times\log(n)$. Il se trouve

Anatoli Karatsuba (1937-2008)

3 LA COMPLEXITÉ DE LA MULTIPLICATION

La multiplication de deux nombres entiers possédant chacun n chiffres, par la méthode apprise à l'école, exige un nombre d'opérations élémentaires de l'ordre de n^2.

```
  123
× 345
  615
 492 .
369 . .
42435
```

On le voit car n^2 est la taille du tableau intermédiaire *(en rouge)* entre les deux lignes horizontales qu'on écrit quand on utilise cette méthode.
Toutes les méthodes connues jusqu'en 1960 avaient une complexité asymptotique en n^2. C'est alors qu'Anatoli Karatsuba (1937-2008), qui connaissait la conjecture formulée par Andreï Kolmogorov selon laquelle on ne pouvait faire mieux que n^2, fit une observation algébrique très simple. Il remarqua qu'un entier de taille $2k$ pouvait s'écrire sous la forme $(a + b10^k)$, où a et b sont deux entiers de k chiffres, et qu'alors le produit de deux entiers de taille $2k$ s'écrit : $(a + b10^k)(c + d10^k) = ac - [(a - b)(c - d) - ac - bd].10^k + bd.10^{2k}$
Lorsqu'on multiplie deux entiers de taille 1 000, cette identité ramène le calcul à trois multiplications au lieu de quatre entre entiers de taille 500 correspondant à un algorithme en n^2. En utilisant ce principe plusieurs fois, c'est-à-dire en ramenant chaque multiplication entre entiers de taille 500 à trois multiplications entre entiers de taille 250, puis chacune des multiplications entre entiers de taille 250 à trois multiplications entre entiers de taille 125, etc., on obtient une multiplication qui utilise un nombre d'opérations élémentaires proportionnel à $n^{\log_2(3)} = n^{1,585}$.
C'est mieux que n^2 même si l'organisation des calculs est devenue plus compliquée.
Comme libérés de la croyance fausse que l'on ne peut faire mieux que n^2 pour multiplier deux entiers de taille n, les mathématiciens ont alors progressé et fait mieux encore que $n^{1,585}$. En 1971, ils sont arrivés à $n\log(n) \times \log(\log(n))$.
Puis tout récemment, David Harvey et Joris van der Hoeven ont publié une méthode en $n \times \log(n)$, ce qui pourrait bien être une méthode optimale du point de vue de la complexité asymptotique.
Malheureusement, les méthodes introduites pour obtenir ces progrès asymptotiques sont de plus en plus complexes, ce qui a pour effet qu'elles ne sont utiles que pour des nombres très grands.
Dans le cas de la méthode en $n \times \log(n)$, la taille des entiers à partir de laquelle il serait intéressant de l'utiliser est si grande que cela n'a aucune chance de se produire avec les nombres que nous utilisons sur Terre. Il s'agit d'un « algorithme galactique ».
Cette situation illustre, plus généralement, que la programmation d'un algorithme peut être plus ou moins compliquée, et que souvent ce sont les algorithmes les plus compliqués qui sont les plus efficaces. Il se produit alors que le meilleur des algorithmes n'est pas meilleur pour les petites valeurs des données et qu'il ne le devient que quand la taille des données dépasse un seuil. La situation est alors la suivante : il faut utiliser une méthode (simple) pour les petites valeurs, et la méthode la plus efficace asymptotiquement à partir du seuil… qui est si grand pour les algorithmes galactiques que jamais cela ne se produit !

cependant que l'algorithme de Fürer ne serait vraiment utile que pour des entiers ayant plus de 2^{64} chiffres. Or $2^{64}=1{,}84\times10^{19}$, et jamais en pratique on n'envisage de telles multiplications, ne serait-ce que parce que mémoriser des nombres de $1{,}84\times10^{19}$ chiffres exigerait des moyens dont personne ne dispose. L'algorithme de Fürer est meilleur que celui de 1971 pour les très grands nombres, mais pour tous les nombres que nous avons à manipuler il ne sert pas. L'algorithme de Fürer est un algorithme galactique !

Après encore quelques étapes de progrès en 2019, le mathématicien australien David Harvey et Joris van der Hoeven, directeur de recherche au CNRS, en France, ont atteint le $n\times\log(n)$ de la conjecture de Schönhage et Strassen. Leur démonstration utilise une technique de transformée de Fourier dans un espace à 1 729 dimensions, qu'il est impossible de décrire ici.

L'examen attentif de l'algorithme de 2019 montre qu'il ne présente un intérêt que pour des entiers ayant un nombre de chiffres supérieur à 2^p, avec $p=17^{2^{9^{12}}}=7{,}13\times10^{38}$, ce qui est bien pire que pour l'algorithme de Fürer et signifie qu'il est inconcevable qu'on l'utilise un jour. Comme celui de Fürer mais en bien plus atroce, l'algorithme de Harvey et van der Hoeven est un algorithme galactique !

Notons que le progrès théorique ne fait pas avancer la conjecture, qu'on ne peut pas avoir une complexité asymptotique meilleure que $n\times\log(n)$ pour la multiplication, car il n'exclut pas que mieux encore soit possible. Le fait d'avoir obtenu des algorithmes galactiques, donc inutilisables, n'est peut-être pas définitif, et David Harvey, l'un des découvreurs de l'algorithme en $n\times\log(n)$ indique : « Nous espérons qu'avec des améliorations supplémentaires, l'algorithme pourra devenir pratique pour les nombres comportant des milliards ou des milliers de milliards de chiffres (10^9 chiffres ou 10^{12} chiffres). Si tel est le cas, il pourrait bien devenir un outil indispensable dans l'arsenal du mathématicien informaticien. »

SOUS-GRAPHES ET TOURS D'EXPOSANTS

Le cas de la multiplication n'est pas le seul : un étonnant et troublant résultat à propos des graphes montre que des situations où il ne peut pas être question de négliger la taille des constantes se produisent à propos de problèmes assez simples.

Un graphe H est un mineur du graphe G si H peut être obtenu en contractant G, c'est-à-dire si H se déduit de G en effectuant un nombre quelconque d'opérations parmi les suivantes : (a) suppression d'un sommet isolé ; (b) suppression d'une arête sans modifier les extrémités ; (c) suppression d'une arête en fusionnant ses deux extrémités en un seul sommet en convenant que si deux des arêtes du graphe créé par cette fusion ont les mêmes extrémités, on n'en garde qu'une (*voir des exemples dans l'encadré 5*).

La question de savoir si H est un mineur de G dans le cas général est un problème NP-complet, ce qui signifie qu'il est très probable qu'il ne peut pas être résolu en temps majoré par un polynôme de la taille des données H et G. Cependant, quand H est fixé, la question de savoir si H est un mineur de G peut être traitée par un algorithme dont la complexité asymptotique est n^2, où n est le nombre de sommets de G. Il se trouve cependant, et c'est totalement fascinant, que la constante qu'il faut introduire pour chaque H fixé, est astronomique dès que H possède plus de trois sommets. Cette constante qui dépend du nombre s de sommets de H s'écrit $2{\uparrow\uparrow}(2{\uparrow\uparrow}(2{\uparrow\uparrow}(s/2)))$ dans la notation de John Conway, notation qui permet de définir des nombres entiers défiant totalement l'imagination (*voir https://en.wikipedia.org/wiki/Conway_chained_arrow_notation*).

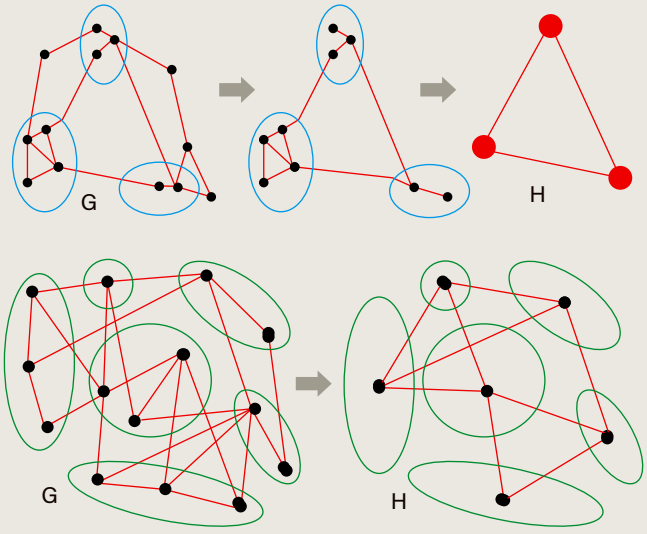
4 L'IA ET LA MULTIPLICATION DES MATRICES

En mathématiques, il n'est pas rare qu'on ne réussisse pas à résoudre un problème car on le croit impossible à résoudre. Puis une fois cette paralysie mentale surmontée, plusieurs progrès successifs se produisent. Ce fut le cas avec la multiplication d'entiers *(voir l'encadré précédent)*. Tout récemment, un autre exemple est venu illustrer cette idée, mais cette fois c'est une intelligence artificielle qui a débloqué la paralysie des chercheurs.
En 2022, la société DeepMind a présenté AlphaTensor, un système d'intelligence artificielle utilisant des réseaux de neurones. Dans le cadre de recherches menées par Alhussein Fawzi, ce système a recherché des algorithmes les plus économes possibles pour la multiplication de deux matrices dans le cas particulier où les coefficients sont des nombres binaires. Pour la multiplication de deux matrices de taille 4 × 4, AlphaTensor a trouvé une façon de procéder qui n'utilise que 47 multiplications entre coefficients, ce qui améliore la meilleure méthode connue qui en exigeait 49.
Les spécialistes de ces problèmes qui ne croyaient pas cela possible se sont remis au travail de leur côté et ont depuis trouvé une méthode en 47 multiplications, différente de celle de l'IA.
De même, AlphaTensor a ramené le nombre de multiplications nécessaires pour des matrices de taille 5 × 5 de 98 à 96, et cette fois encore les humains stimulés par ce résultat ont su en trouver une meilleure en 95 multiplications. Il n'est pas certain que ces progrès puissent vraiment être utiles en pratique. En matière de complexité des algorithmes, il apparaît qu'avec l'IA ou sans, on progressera encore longtemps même pour les problèmes où, à tort, on pense tout savoir.
Sur l'illustration ci-dessous, on a représenté la multiplication traditionnelle de matrices.

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \begin{bmatrix} j & k & l \\ m & n & o \\ p & q & r \end{bmatrix} = \begin{bmatrix} (aj + bm + cp) & (ak + bn + cq) & (al + bo + cr) \\ (dj + em + fp) & (dk + en + fq) & (dl + eo + fr) \\ (gj + hm + ip) & (gk + hn + iq) & (gl + ho + ir) \end{bmatrix}$$

5 LA MAUDITE CONSTANTE EN THÉORIE DES GRAPHES

En théorie des graphes, on dit que le graphe H est un mineur du graphe G si on peut retrouver la structure de H dans G quand on supprime des éléments de G (arêtes, ou sommets) et qu'on fusionne certaines paires de sommets. Deux exemples sont représentés sur la figure ci-contre.
Bien que la notion de mineur soit naturelle et semble assez simple ; savoir si un H est un mineur de G exige un calcul d'une grande complexité sauf pour des H de très petite taille. Si H est fixé, on connaît un algorithme permettant de savoir si oui ou non H est un mineur de G. L'algorithme est d'une complexité asymptotique en n^2 où n est le nombre de sommets de G, ce qui signifie que le temps de calcul est inférieur à $C \times n^2$ pour une certaine constante C qui dépend de H. Il se trouve cependant que la constante C est tellement énorme qu'en réalité, dès que H possède plus de 3 sommets, il est impossible d'utiliser l'algorithme… qui est un algorithme galactique.



Indiquons ce que cela signifie pour $s=4$. La constante à écrire devant le n^2 donnant la complexité de l'algorithme connu est une tour d'exponentielles $2^{2^{2^{2}}}$ à 655 536 étages.

Pour mesurer à quel point ce nombre est grand, sachez que dans une telle tour d'exponentielles, quand il y a cinq « 2 », le nombre représenté possède déjà plus de 19 000 chiffres décimaux. Il semble impossible de se faire une idée du nombre de chiffres de cette tour d'exponentielles quand elle possède 655 536 étages, ce qui n'est pourtant que la constante qu'il faut utiliser pour $s=4$.

En clair, pour un graphe H de plus de 3 sommets, l'algorithme connu en n^2 tiré des articles théoriques et permettant de savoir si H est un mineur de G est un algorithme gravement galactique ! Comme dans le cas de la multiplication, cela n'interdit pas dans le futur de trouver peut-être de meilleurs algorithmes.

LE VOYAGEUR DE COMMERCE

Voici un dernier exemple particulièrement étrange de situation où un progrès théorique en apparence intéressant n'a pas de véritable intérêt pratique.

Le problème du voyageur de commerce consiste à trouver comment passer une fois exactement par chaque sommet d'un graphe dont chaque arête a une longueur connue en empruntant le chemin le plus court possible et en revenant au point de départ. Pour un tel parcours, on parle de circuit hamiltonien optimal. Dans le cas de points d'un espace métrique, c'est-à-dire quand la distance entre les points du graphe vérifie l'inégalité triangulaire $d(a,c) \leq d(a,b)+d(b,c)$, l'algorithme de Nicos Christofides, découvert en 1976, donne une solution approchée intéressante. L'algorithme ne trouve pas nécessairement le meilleur circuit, mais en construit un dont la longueur ne dépasse pas plus de 50% celle du meilleur circuit.

UN PROGRÈS MINIME MAIS IMPORTANT

En pratique, on utilise des algorithmes qui, pour la majorité des graphes, sont meilleurs que l'algorithme de Christofides, mais dont on n'a pas de garantie qu'ils font toujours mieux. On a donc le choix entre un algorithme qui ne sera pas mauvais de plus de 50% avec certitude, ou des algorithmes qui sont souvent meilleurs sans que cela soit certain. Très étrangement, en 2021, Anna Karlin, de l'université de Washington, aux États-Unis, et ses collègues ont proposé un algorithme qui améliore de manière certaine l'algorithme de Christofides, perfectionnement qu'on recherchait depuis plus de quarante ans. C'est un exploit mathématique étonnant, qui a été salué par la communauté des spécialistes.

Le nouvel algorithme assure que le circuit trouvé aura une longueur d'au plus $(3/2-10^{-32})\times L$, où L est la longueur du meilleur circuit. C'est une amélioration incontestable de l'algorithme de Christofides, mais elle est vraiment minime ! Comme le nouvel algorithme est plus compliqué et que le gain qu'il donne est infinitésimal, il en résulte que personne ne l'utilisera pour résoudre des problèmes réels ! L'intérêt mathématique du nouvel algorithme est qu'il montre que la constante 3/2 pour un algorithme n'est pas une borne infranchissable. Peut-être est-ce seulement le premier pas vers d'autres améliorations ? ■

BIBLIOGRAPHIE

A. Fawzi *et al.*, **Discovering faster matrix multiplication algorithms with reinforcement learning,** *Nature*, 2022.

D. Harvey et J. van der Hoeven, **Integer multiplication in time O(*n* log *n*),** *Annals of Mathematics*, 2021.

A. Karlin *et al.*, **A (slightly) improved approximation algorithm for metric TSP,** *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, 2021.

R. Lipton et K. Regan, **David Johnson : Galactic Algorithms, in People, Problems, and Proofs,** Springer, 2013.

D. Johnson, **The NP-completeness column : An ongoing guide,** *Journal of Algorithms*, 1987.