Multiplication binaire et factorisation

Position du problème

On cherche un système dynamique discret qui, étant donné un entier non-premier, trouve un de ses diviseurs non-triviaux. On se base sur la multiplication binaire. En effet, on observe qu'une matrice à coefficients dans $\{0, 1\}$ peut s'interpréter comme une multiplication binaire si ses lignes non-nulles sont égales entre elles, comme illustré dans la figure 1. Les colonnes non-nulles sont alors aussi égales entre elles, et les deux facteurs de la multiplication sont alors lisibles, en base 2, sur les lignes non-nulles pour l'un, et de bas en haut sur les colonnes non-nulles pour l'autre. Si on pose leur multiplication, toujours en base 2, on retrouve en effet la matrice de départ décalée d'un cran par ligne vers la gauche (cf. Figure 1).

1	0	1	1
0	0	0	0
1	0	1	1
1	0	1	1

					1	0	1	1	$\times$
					1	0	1	1	1
+				0	0	0	0	.	0
+			1	0	1	1	.	.	1
+		1	0	1	1	.	.	.	1
	1	0	0	0	1	1	1	1	

FIGURE 1 – La matrice de gauche s'interprète comme la multiplication de 11 (1011 horizontalement) par 13 (1101 verticalement, de haut en bas).

Par ailleurs, étant donné une matrice M à coefficients dans $\{0, 1\}$, on pose $\sigma(M)$ l'entier obtenu en sommant ses lignes décalées (cf. Figure 2). Si M représente une multiplication, σ est par construction le produit.
L'idée est donc de partir d'une matrice à coefficients dans $\{0, 1\}$ pour laquelle σ vaut le nombre qu'on cherche à factoriser, et de lui appliquer des transformations qui conservent σ jusqu'à atteindre une matrice qui représente une multiplication.

$$M = \begin{matrix} 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \end{matrix} \qquad \sigma(M) = \begin{array}{cccccccc} & & & & 1 & 1 & 0 & 1 \\ + & & & 0 & 0 & 1 & 1 & . \\ + & & 1 & 0 & 0 & 1 & . & . \\ \hline & & 1 & 1 & 0 & 1 & 1 & 1 \end{array} = 55$$

FIGURE 2 – Exemple de calcul de σ.

$$\mathbf{1} \quad \underset{\text{R2}}{\overset{\text{R1}}{\rightleftarrows}} \quad \mathbf{0} \qquad\qquad 0 \quad \underset{\text{R2}}{\overset{\text{R1}}{\rightleftarrows}} \quad 1$$

$$\begin{matrix} 0 \\ \mathbf{1} \end{matrix} \quad \underset{\text{R4}}{\overset{\text{R3}}{\rightleftarrows}} \quad \begin{matrix} 1 \\ \mathbf{0} \end{matrix} \qquad\qquad \begin{matrix} 0 \\ 1 \end{matrix} \quad \underset{\text{R4}}{\overset{\text{R3}}{\rightleftarrows}} \quad \begin{matrix} 1 \\ 0 \end{matrix}$$

$$\mathbf{1} \quad 0 \quad \underset{\text{R6}}{\overset{\text{R5}}{\rightleftarrows}} \quad \mathbf{0} \quad 1 \qquad\qquad 1 \quad 0 \quad \underset{\text{R6}}{\overset{\text{R5}}{\rightleftarrows}} \quad 0 \quad 1$$

FIGURE 3 – Transformations locales préservant σ.

Un algorithme

Soit n le nombre de chiffres en base 2 du nombre composé N dont on cherche un diviseur non-trivial. On se donne une matrice M de taille $n-1 \times \lceil \frac{n}{2} \rceil$ telle que $\sigma(M) = N$. Pour tous a et b différents de 1 et N, et tels que $N = ab$, a ou b est de taille inférieure à $\lceil \frac{n}{2} \rceil$ et l'autre est de taille inférieure à $n-1$, on peut donc représenter leur multiplication sur une matrice de la taille de M. On l'initialise en recopiant les $n-1$ bits de poids faible de N sur la première ligne, et avec un 1 en position $(2,1)$ et des zéros partout ailleurs. On vérifie aisément que $\sigma(M) = N$.
Formellement, on pose $\mathcal{L} = [\![1, n-1]\!] \times [\![1, \lceil \frac{n}{2} \rceil]\!]$ l'ensemble des cases de la matrice, et $\mathcal{E} = \{0,1\}^{\mathcal{L}}$ l'ensemble des configurations. Pour tout $x \in \mathcal{E}$, $(i,j) \in \mathcal{L}$, on note $x_{i,j}$ le coefficient (i,j) de la matrice x.
On se donne également un ensemble $\Gamma \subset \mathcal{E}^{\mathcal{E} \times \mathcal{L}}$ de transformations locales qui conservent σ. Elles sont représentées sur la figure 3, et s'appliquent aux coefficients indiqués en gras sous certaines conditions.
Une formulation équivalente de l'égalité des lignes non-nulles est qu'un coefficient situé sur la même ligne qu'un 1 et sur la même colonne qu'un 1 vaut nécessairement 1 dans une matrice qui représente une multiplication.

0	1	0	0	1	1	0	$\boxed{0}$
0	$\boxed{0}$	0	0	1	1	0	1
0	$\boxed{0}$	0	0	1	1	0	1
0	0	0	0	0	0	0	0
0	$\boxed{0}$	0	0	1	1	0	1

FIGURE 4 – Exemple de situation bloquée si on interdit les mouvements ne déplaçant pas les zéros menacés (encadrés) : aucune des règles de la Figure 3 ne s'applique à eux.

On va donc appliquer aléatoirement des règles conservant σ, en privilégiant les mouvements qui retirent les zéros de telles cases. Dans la suite, on dira qu'une case est *sûre* s'il n'y a pas de 1 sur la même ligne ou s'il n'y en a pas sur la même colonne, et *dangereuse* s'il y a à la fois un 1 sur la même ligne et un sur la même colonne.
Pour diriger les zéros vers les cases sûres, on voudrait n'effectuer que des mouvements qui sortent un zéro d'une case dangereuse, mais cette restriction est trop forte : il existe alors des situations bloquées, la figure 4 en donne un exemple. Il est donc nécessaire d'autoriser au moins certains mouvements qui ne sont pas directement utiles en ce sens, pour permettre d'autres mouvements par la suite. De même, on pourrait vouloir restreindre les mouvements des zéros vers des cases dangereuses, de la même façon que pour le problème des n reines. Cependant, il se forme alors des blocs stables de 1 : les zéros qui viennent remplacer un 1 du bord du bloc en sont rapidement éjectés (les seules cases sûres proches sont à l'extérieur du bloc).
On prend donc le parti de ne pas restreindre les mouvements des zéros vers des cases dangereuses, et on se donne une probabilité d'agitation p : un mouvement valide mais qui ne sort pas un zéro d'une case dangereuse est effectué avec une probabilité p, alors qu'un mouvement valide qui sort un zéro d'une case dangereuse est toujours effectué.
À chaque pas de temps, on choisit donc une case de $\mathcal{L}$ et une règle de Γ au hasard, et on applique éventuellement la règle à la case. À titre d'exemple, si on choisit, dans une configuration x, la case (i, j) et la règle **R1**, la nouvelle configuration est

$$\mathbf{R1}(x,(i,j)) = \begin{cases} x' & \text{si} \begin{cases} x_{i,j} = 1 \text{ et } x_{i+1,j+1} = 0 \\ \text{et} \begin{cases} \text{soit } (i+1,j+1) \text{ est une case dangereuse} \\ \text{soit } \mathcal{B}(p) = 1 \end{cases} \end{cases} \\ x & \text{sinon} \end{cases}$$

avec x' la configuration obtenue à partir de x en échangeant les valeurs de $x_{i,j}$ et $x_{i+1,j+1}$, et $\mathcal{B}(p)$ une variable aléatoire valant 1 avec une probabilité p et 0 avec une probabilité $1-p$.
On applique une règle si au moins un des zéros concernés est sur une case dangereuse : ainsi

$$\mathbf{R3}(x,(i,j)) = \begin{cases} x'' & \text{si} \begin{cases} x_{i,j} = 1 \text{ et } x_{i-1,j} = x_{i,j+1} = 0 \\ \text{et} \begin{cases} \text{soit } (i-1,j) \text{ est une case dangereuse} \\ \text{soit } (i,j+1) \text{ l'est} \\ \text{soit } \mathcal{B}(p) = 1 \end{cases} \end{cases} \\ x & \text{sinon} \end{cases}$$

avec x'' la configuration obtenue à partir de x en changeant les valeurs de $x_{i,j}$, $x_{i-1,j}$ et $x_{i,j+1}$.

Du fait de l'existence de mouvements spontanés, il n'y a pas de point fixe à proprement parler : on est contraint de vérifier après chaque mouvement si la matrice représente une multiplication valide, et d'arrêter les mises à jour le cas échéant.

Performances et observations

Les performances varient grandement selon la décomposition en facteurs premiers du nombre à factoriser. Ainsi, les nombres pour lesquelles les matrices solutions du problème comportent peu de 1 semblent bien plus difficiles : alors que l'algorithme trouve un diviseur de 493 (17×29) en 9×10^5 pas de temps en moyenne, il lui faut 10^9 étapes pour factoriser 362 (2×181) ! Ceci est dû au fait que la densité de 1 varie très peu au cours du temps (elle oscille autour d'une densité moyenne, et les solutions pour lesquelles la densité finale en est éloignée sont difficiles à trouver).

De plus, les conditions que l'on s'est donné pour appliquer les règles tendent à chasser les zéros menacés, mais rien ne pousse particulièrement les 1 isolés à se déplacer. Il est donc relativement difficile de trouver les positions pour lesquelles les 1 sont tous à un endroit précis de la matrice : ainsi, pour les nombres de la forme $2^k p$ avec p premier, la matrice aura tous ses 1 sur la même ligne (la k-ième) ou bien sur la même diagonale. Ainsi, dans les cas "faciles" tels que 493, une solution est atteinte environ 40 fois plus vite avec $p = 0.05$ qu'avec $p = 1$ (cas où le système applique des règles valides sans tenir compte du caractère dangereux ou sûr des cases), mais dans les cas difficiles tels que 362, les performances sont plutôt meilleures avec la marche

aléatoire $p = 1$. Plus généralement, l'évolution du système n'est dirigée que faiblement, notamment parce qu'on a décidé de ne pas restreindre les déplacements de zéros vers des cases dangereuses et parce que le système n'a pas de mémoire.

Le fait que les 1 isolés ne soient pas encouragés à se déplacer est d'autant plus gênant qu'on a choisi une taille de matrice nécessairement trop grande pour les facteurs que l'on va trouver. En effet, on a pris une matrice de taille $n - 1 \times \lceil \frac{n}{2} \rceil$ pour un nombre de taille n afin d'assurer de pouvoir trouver n'importe quel diviseur non trivial, mais la somme des longueurs des deux facteurs que l'on va trouver est au plus $n+1$. Par conséquent, tous les 1 d'une matrice solution sont contenus dans un rectangle dont l'un des sommets est le coin supérieur droit et dont le périmètre est limité : s'il y a un 1 dans le coin inférieur droit, il ne peut y en avoir aucun dans toute la moitié gauche de la matrice ! Si, dans une exécution de l'algorithme, aucun 1 n'apparaît typiquement dans le quart inférieur gauche pour p assez petit, il y en a fréquemment à la fois dans le coin supérieur gauche et inférieur droit, ce qui fait perdre beaucoup de temps.

Une amélioration

On peut donc envisager l'amélioration suivante : plutôt que d'utiliser une grande matrice, on suppose connue la taille des diviseurs. Si n est la taille du nombre à factoriser, il y a au plus $2n-1$ tailles de matrices possibles à essayer car la somme des tailles des diviseurs vaut nécessairement n ou $n + 1$ et on ignore les matrices dont une dimension vaut 1 car on cherche des diviseurs non-triviaux. On lance donc en parallèle l'algorithme sur les $2n - 1$ matrices différentes, et on arrête l'exécution dès qu'une solution est trouvée pour l'une des tailles. Quitte à supposer impair le nombre à factoriser, on impose de plus qu'il y ait un 1 à chacun des quatre coins (c'est une condition nécessaire pour qu'une matrice dont les dimensions sont exactement les tailles de deux facteurs soit solution : chaque facteur commence par un 1 car les zéros à gauche ne sont pas significatifs, et finissent par un 1 car les diviseurs d'un nombre impair sont impairs).

Il apparaît que l'exécution de l'algorithme est significativement plus rapide qu'avec la grande matrice. Les nombres dont l'un des facteurs est très petit deviennent très faciles à traiter, car la matrice de la bonne taille est très petite, et le calcul y aboutit donc rapidement. Les nombres les plus difficiles à traiter deviennent, comme pour les algorithmes classiques de factorisation, les produits de deux nombres premiers de même taille : en effet, la seule

matrice pour laquelle l'algorithme peut aboutir est alors de taille maximale. Même dans ce cas, l'exécution est significativement plus rapide car on ne perd plus de temps à cause des 1 situés dans une zone inutile de la matrice. On gagne ainsi un facteur 100 sur la factorisation de 329 (7×47), largement de quoi justifier de lancer simultanément une quinzaine d'exécutions.