Multiplication binaire et factorisation

Introduction

J'ai effectué mon stage dans les équipes Algorille et Maia au Loria (Nancy). Il consistait en l'élaboration de systèmes dynamiques discrets répondant à deux problèmes algorithmiques : le problème des n reines et la factorisation d'entiers. L'objectif était d'obtenir des systèmes relativement simples, tels que des automates cellulaires ou des systèmes multi-agents [2], qui convergent vers une solution du problème considéré [1]. Plus précisément, mon travail a été de concevoir de tels systèmes, puis de programmer un simulateur afin de les tester et d'envisager des améliorations.

Le problème des n reines

1 Position du problème

On se place sur un échiquier de taille $n \times n$, sur lequel n reines sont disposées. On dit qu'une reine est *menacée* si elle se trouve sur la même ligne, colonne ou diagonale qu'une autre reine. On cherche à placer les n reines de sorte qu'aucune reine ne soit menacée. On va chercher, en partant d'une situation initiale quelconque, à déplacer les reines de sorte à atteindre une position dans laquelle aucune d'elles n'est menacée. Pendant ces déplacements, on va permettre à chaque case de contenir un nombre quelconque de reines simultanément, et aussi d'être marquée par des signaux provenant d'un nombre quelconque de directions, qui traduisent la présence d'une reine dans la direction d'où ils proviennent.

On cherche une procédure de mise à jour locale telle qu'à chaque pas de temps, les états de certaines cases sont modifiés à partir d'informations sur les cases voisines : création et transmission de signaux, déplacements de reines... L'objectif est que les signaux traduisent au mieux la présence de reines dans la direction d'où ils viennent, et que les reines menacées se déplacent au cours du temps pour chercher des cases sûres. On cherche également à ce que les reines ne bougent plus une fois qu'une solution est atteinte. On s'astreint à ne déplacer les reines que d'au plus une case par pas de temps, de sorte que la règle de mise à jour puisse rester purement locale.

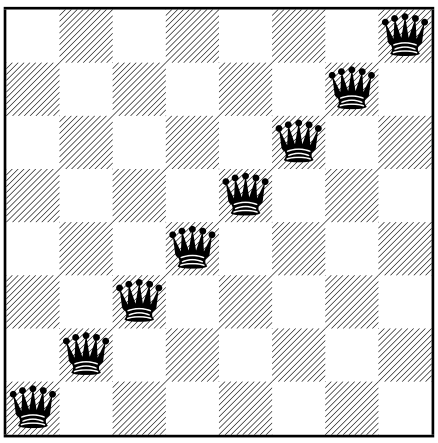

Figure 1 – Les reines sont toutes menacées, mais aucun déplacement d'une case ne conduit à une case sûre.

On voudrait ne déplacer les reines que sur des cases sûres, mais c'est trop restrictif : il existe des positions qui ne sont pas solution du problème mais où toutes les reines sont entourées de cases menacées (cf. Figure 1). En revanche, permettre aux reines de se déplacer sur des cases dangereuses sans utiliser les informations locales dont elles disposent donne des mouvements

trop aléatoires pour qu'on puisse espérer atteindre une solution en un temps raisonnable. On fixe donc une probabilité $\varepsilon \in [0,1]$, qui correspond à la probabilité pour une reine qui envisage un déplacement vers une case dangereuse de l'effectuer, alors qu'un déplacement vers une case sûre est systématique. En fixant ε assez petit (de l'ordre de 0.01 par exemple), on privilégie les déplacements vers des cases sûres tout en se donnant quand même la possibilité de sortir de situations bloquées.

2 Formalisation

2.1 Représentation du système

On représente l'échiquier par l'ensemble $\mathcal{L} = [\![0, n-1]\!]^2$. On pose $\mathcal{P} = \{0,1\}$ et $\mathcal{D} = \{NW, N, NE, W, E, SW, S, SE\}$ l'ensemble des huit directions. L'état d'une cellule étant déterminé par le nombre de reines présentes et la présence ou l'absence d'un signal en provenance de chaque direction, on note $\mathcal{Q} = \mathcal{P}^8 \times \mathbb{N}$ l'ensemble des états. Une *configuration* est un élément de $\mathcal{E} = \mathcal{Q}^{\mathcal{L}}$. Pour toute configuration x, et toute cellule $c \in \mathcal{L}$, on note $x_c = (s_{x,c}, n_{x,c})$ l'état de la cellule c dans la configuration x, où $n_{x,c}$ est le nombre de reines sur la cellule c dans la configuration x, et $s_{x,c} = (s_{x,c,N}, s_{x,c,NE}, s_{x,c,E}, \ldots, s_{x,c,NW})$ code la présence ou l'absence de signaux venant de chaque direction : $s_{x,c,d}$ vaut 1 si dans la configuration x, la cellule c contient un signal provenant de la direction d, et 0 sinon.
Par souci de lisibilité, on omettra les indices x de configuration partout où on pourra le faire sans ambiguïté et on parlera de l'état (s_c, n_c) d'une cellule c.
On pose $\mathcal{N} \subset \mathcal{L}^2$ l'ensemble des paires de cellules adjacentes :

$$(c, c') \in \mathcal{N} \iff \left\|(c - c')\right\|_\infty = 1$$

À chaque pas de temps, on choisit au hasard une paire de cases adjacentes et on les met à jour l'une par rapport à l'autre. On se donne donc $(u_t)_{t\in\mathbb{N}} \in \mathcal{N}^{\mathbb{N}}$ la suite des paires de cellules mises à jour : pour tout $t \in \mathbb{N}$, la paire de cellules mises à jour à l'étape t est u_t. Ce sont les valeurs d'une suite de variables aléatoires indépendantes à distribution uniforme sur $\mathcal{N}$. On définit de même $(x_t)_{t\in\mathbb{N}}$ la suite des configurations prises par le système au cours du temps. Initialement, aucun signal n'est présent et les n reines sont réparties aléatoirement sur l'échiquier, selon une loi uniforme (de façon indépendante : notamment plusieurs reines peuvent se trouver initialement sur la même case). On cherche à définir une fonction de mise à jour globale $\mathcal{F} : \mathcal{E} \times \mathcal{N} \to \mathcal{E}$ qui à une configuration associe une configuration qui lui succède selon la procédure de la partie précédente : $\mathcal{F}(x_t, u_t) = x_{t+1}$.

2.2 Choix des cases à mettre à jour

On souhaite choisir une paire de cases adjacentes à mettre à jour, avec la même probabilité de choisir chaque paire de cases. On observe qu'il y a $n(n-1)$ paires de cases voisines verticalement, $n(n-1)$ horizontalement et $(n-1)^2$ selon chaque diagonale, soit $(4n-2)(n-1)$ paires de cases adjacentes au total. On applique donc la procédure de sélection suivante :

- On choisit selon une loi uniforme $N \in [\![0, (4n-2)(n-1)-1]\!]$, on en déduit $A \in [\![0, 4n-3]\!]$ et $B \in [\![0, n-2]\!]$ le quotient et le reste de la division euclidienne de N par $(n-1)$.
- Si $A < n$, on prend les cases (B, A) et $(B+1, A)$.
- Si $A = n + i, 0 \leq i < n$, on prend les cases (i, B) et $(i, B+1)$.
- Si $A = 2n + i, 0 \leq i < n-1$, on prend (i, B) et $(i+1, B+1)$.
- Sinon, $A = 3n - 1 + i$ et $0 \leq i < n-1$, et on prend $(i, B+1)$ et $(i+1, B)$.

La fonction qui à N associe une paire de cases adjacentes étant bijective, on a bien la même probabilité de choisir chaque paire de cases.

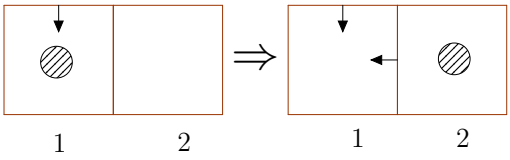

FIGURE 2 – La reine présente en 1, menacée depuis le nord, se déplace en 2 qui est une case sûre, puis les signaux sont mis à jour : il y a une reine en 2, donc un signal venant de l'est apparaît en 1.

2.3 Mise à jour : les déplacements

On se donne deux cases adjacentes c et c' à mettre à jour. On commence par envisager des déplacements de reine entre c et c'. Le principe général est qu'une reine menacée essaie de quitter sa case lorsqu'elle est mise à jour. Ce déplacement réussit ou non en fonction de l'état de l'autre case mise à jour, avec une part d'aléatoire. La procédure est la suivante :

- Si une reine est présente en case c, alors que la case c contient un signal, elle essaie de se déplacer vers la case c'. De même, si plusieurs reines sont présentes en c, elles sont menacées et l'une d'entre elles tente de se déplacer vers la case c'.
- S'il n'y a pas de signal en c' sauf éventuellement un venant de c (qui est peut-être dû à la présence de la reine qui cherche à bouger), et qu'il n'y a pas d'autre reine en c', la case est considérée comme sûre et le déplacement a lieu (cf. Figure 2). Sinon, la case est dangereuse, et le déplacement a lieu seulement avec une probabilité ε.
- Symétriquement, une reine de c' peut se déplacer vers c selon la même procédure. Les deux déplacements se font de manière simultanée afin que l'ordre des deux cases choisies pour la mise à jour n'ait pas d'importance.

Formalisons cette procédure à l'aide de quelques fonctions auxiliaires :

- On pose $\textsc{safestay} : \mathcal{L} \to \{0,1\}$ qui indique si une reine peut rester sur une case donnée :

$$\textsc{safestay}(c) = \begin{cases} 1 & \text{si } n_c \leq 1 \text{ et } \forall d \in \mathcal{D}, s_c(d) = 0 \\ 0 & \text{sinon} \end{cases}$$

- On pose $\textsc{safego} : \mathcal{L} \times \mathcal{D} \to \{0,1\}$ qui indique si une reine peut se déplacer de façon sûre vers une certaine case en venant d'une certaine direction :

$$\textsc{safego}(c, d_{\text{from}}) = \begin{cases} 1 & \text{si } n_c = 0 \text{ et } \forall d \in \mathcal{D} \setminus \{d_{\text{from}}\}, s_c(d) = 0 \\ 0 & \text{sinon} \end{cases}$$

  En toute rigueur, SAFESTAY et SAFEGO devraient prendre aussi la configuration en argument : on l'omet pour plus de lisibilité.
- On pose $\delta : \mathcal{N} \to \mathcal{D}$ telle que $\delta(c_{\text{from}}, c_{\text{to}})$ soit la direction de c_{from} vue de c_{to}. Plus formellement :

$$\delta(c_{\text{from}}, c_{\text{to}}) = \begin{cases} N & \text{si } c_{\text{to}} - c_{\text{from}} = (0,1) \\ NE & \text{si } c_{\text{to}} - c_{\text{from}} = (1,1) \\ \cdots & \\ W & \text{si } c_{\text{to}} - c_{\text{from}} = (-1,0) \\ NW & \text{si } c_{\text{to}} - c_{\text{from}} = (-1,1) \end{cases}$$

- On peut alors définir MOVE : $\mathcal{N} \to \{0,1\}$ qui traduit la procédure décrite ci-dessus : $\text{MOVE}(c_{\text{from}}, c_{\text{to}})$ vaut 1 si une reine se déplace de c_{from} à c_{to} et 0 sinon. Formellement :

$$\text{MOVE}(c_{\text{from}}, c_{\text{to}}) = \begin{cases} 0 & \text{si } \text{SAFESTAY}(c_{\text{from}}) = 1 \\ 1 & \text{si : } \begin{cases} \text{SAFESTAY}(c_{\text{from}}) = 0 \\ \text{et } n_{c_{\text{from}}} \geq 1 \\ \text{et } \text{SAFEGO}(c_{\text{to}}, \delta(c_{\text{from}}, c_{\text{to}})) = 1 \end{cases} \\ \mathcal{B}(\varepsilon) & \text{sinon} \end{cases}$$

  où $\mathcal{B}(\varepsilon)$ est une variable aléatoire valant 1 avec une probabilité ε et 0 avec une probabilité $1 - \varepsilon$. Comme précédemment, on omet de passer la configuration en argument à MOVE tant qu'il n'y a pas d'ambiguïté.

On peut maintenant écrire $\mathcal{F}_{\text{move}} : \mathcal{E} \times \mathcal{N} \to \mathcal{E}$ qui modifie la configuration par d'éventuels déplacements de reine entre les cases mises à jour :

$$\mathcal{F}_{\text{move}}(x, (c_1, c_2)) = \begin{cases} x_{c_1 \to c_2} & \text{si } \text{MOVE}(c_1, c_2) = 1 \text{ et } \text{MOVE}(c_2, c_1) = 0 \\ x_{c_2 \to c_1} & \text{si } \text{MOVE}(c_1, c_2) = 0 \text{ et } \text{MOVE}(c_2, c_1) = 1 \\ x & \text{sinon} \end{cases}$$

où $x_{a \to b}$ désigne la configuration obtenue à partir de x en ôtant 1 à n_a et en ajoutant 1 à n_b.

2.4 Mise à jour : les signaux

Après avoir effectué d'éventuels déplacements de reine, on met à jour les signaux de c et c'.

- Un signal en c' provenant de la direction de c apparaît ou reste présent s'il y a une reine en c (cf. Figure 2), ou s'il y a un signal en c dans la même direction (intuitivement, il y a une reine plus loin dans la direction de c) (cf. Figure 3). Sinon, rien n'indique plus la présence d'une reine dans cette direction : le signal disparaît s'il était présent (cf. Figure 4).
- Symétriquement, le signal en c provenant de c' est mis à jour de la même façon.
- Les autres signaux ne sont pas modifiés.

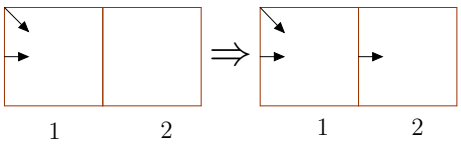

FIGURE 3 – Le signal de la cellule 1 venant de l'ouest est transmis à la cellule 2.

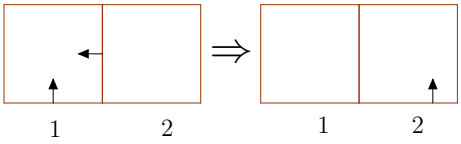

FIGURE 4 – Le signal en 1 venant de l'est disparaît.

Formellement, on pose, pour tout $d \in \mathcal{D}$, $s^{c'}_{x,c,d}$ le signal de la case c se rapportant à la direction d, mis à jour par rapport à c' selon la règle :

$$s^{c'}_{x,c,d} = \begin{cases} s_{x,c,d} & \text{si } d \neq \delta(c', c) \\ 1 & \text{si } d = \delta(c', c) \text{ et } n_{x,c'} \geq 1 \\ s_{x,c',d} & \text{sinon} \end{cases}$$

On va s'en servir pour définir une fonction $\mathcal{F}_{\text{sign}} : \mathcal{E} \times \mathcal{N} \to \mathcal{E}$ de mise à jour des signaux : $\mathcal{F}_{\text{sign}}(x, (c_1, c_2)) = x'$ avec :

$$\begin{cases} x'_{c_1} = (s^{c_2}_{x,c_1}, n_{x,c_1}) \\ x'_{c_2} = (s^{c_1}_{x,c_2}, n_{x,c_2}) \\ \quad x'_c = x_c \qquad \text{pour tout } c \notin \{c_1, c_2\} \end{cases}$$

On peut alors définir la fonction de mise à jour :

$$\mathcal{F}(x_t, u_t) = \mathcal{F}_{\text{sign}}(\mathcal{F}_{\text{move}}(x_t, u_t), u_t)$$

On remarque que lors de la mise à jour d'une case, la nouvelle valeur du signal qu'on modifie ne dépend pas de l'ancienne valeur, mais seulement de la case voisine : les signaux sont transmis localement de manière instantanée mais les cases n'ont pas de mémoire. Ainsi, lorsqu'une reine quitte une rangée de cases, les signaux dans cette direction n'arrivent plus et l'information de l'absence de cette reine est propagée.
L'asynchronisme de la mise à jour est utile : si on mettait les cases à jour de façon synchrone, deux reines sur la même ligne, colonne ou diagonale recevraient en même temps l'information, et pourraient se déplacer toutes les deux en même temps alors que le comportement souhaité est qu'une seule des deux reines se déplace.

3 Expérimentations

Ajouter un bref contexte expérimental (type ordi, OS, langage développement)

Lancé à partir d'une situation de départ aléatoire (la case de départ de chacune des n reines est choisie selon une loi uniforme de façon indépendante), on observe que le système finit par atteindre un point fixe où chaque reine est sur une case sans signal. La position des reines est alors solution du problème des n reines.

3.1 Comportement qualitatif

On observe en général un régime transitoire sur les premiers milliers de pas de temps, pendant lesquels les signaux ne se sont pas encore propagés sur tout l'échiquier, donc pendant lesquels les mouvements des reines sont très libres. Les mouvements des reines sont rares par la suite : une fois les signaux convenablement propagés, il y a très peu de cases sûres sur l'échiquier, voire aucune. De temps en temps, une reine se déplace vers une case dangereuse, ce qui libère certaines cases ; des déplacements vers des cases devenues sûres ont lieu puis on retrouve une situation de blocage, et ainsi de suite jusqu'à ce qu'une solution soit trouvée.

3.2 Effets de ε

Le paramètre ε a des effets multiples. Plus ε est grand, plus le système sort rapidement de situations bloquées où aucun déplacement vers une case sûre n'est possible. En revanche, plus il est petit, plus le système a tendance à effectuer tous les déplacements possibles vers des cases sûres avant de déplacer une reine vers une case dangereuse, ce qui peut empêcher de passer à côté d'une solution. En effet, si une solution est accessible sans avoir besoin de déplacer de reine vers une case dangereuse, plus ε est grand, plus le risque de déplacer une reine vers une case dangereuse et d'avoir peu de chances de trouver rapidement cette solution est important.

Enfin, comme il y a typiquement très peu de cases sûres, ε contrôle également la fréquence des déplacements : plus il est petit, plus les signaux ont de temps pour se propager sur l'échiquier avant le prochain déplacement de reine. Ceci implique que plus ε est petit, plus les reines se déplacent en se basant sur des informations à jour. Indirectement, ε contrôle donc également le degré d'asynchronisme à retards du système.
Le choix de ε pour minimiser le temps mis par le système pour trouver une solution relève donc d'un compromis entre ces différents effets : on le voudrait petit pour privilégier les déplacements sûrs et déplacer les reines à partir d'informations valides, mais grand pour que les déplacements soient suffisamment fréquents.
La Figure 5 représente la variation du temps de convergence (le nombre de mises à jour avant d'atteindre une solution) et du nombre de déplacements effectifs de reine en fonction de la probabilité ε, dans le cas $n = 8$. Les valeurs sont des moyennes sur 100 échantillons, sauf les deux points extrémaux où l'on s'est restreint à 30 échantillons pour des raisons de temps de calcul. On observe un optimum de ε pour ce qui est du temps de convergence, autour de 0.01, tandis que le nombre de déplacements croît avec ε sur les valeurs considérées.

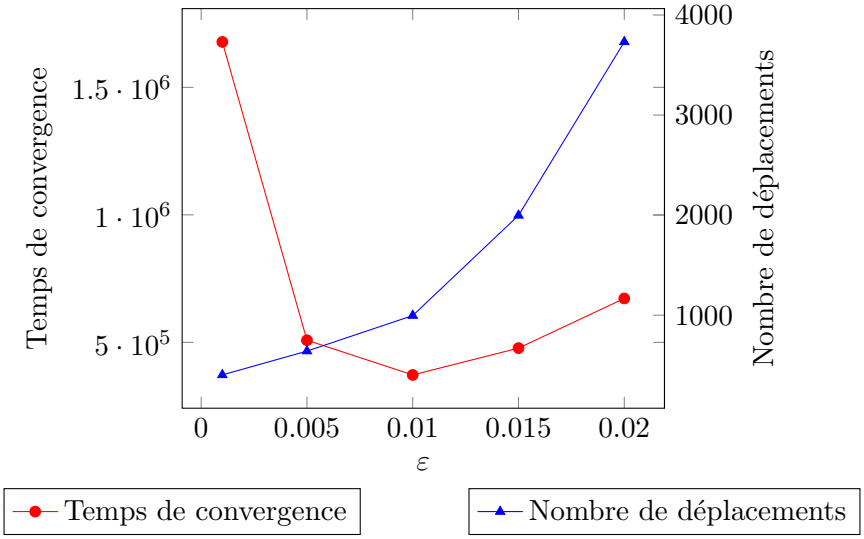

FIGURE 5 – Temps de convergence moyen et nombre moyen de déplacements en fonction de ε pour $n = 8$.

La monotonie apparente du nombre de déplacements en fonction de ε n'est pas étonnante : diminuer ε revient à augmenter la qualité des déplacements (plus souvent vers des cases sûres, avec des informations plus à jour) quitte à ce qu'ils aient lieu moins fréquemment.
Ce modèle ne fait pas de distinction entre les cases dangereuses : on aurait pu envisager, par exemple, de toujours autoriser les déplacements vers des cases portant moins de signaux que la case de départ. Cependant, nos tests ont montré que cette modification a tendance à détériorer la vitesse de convergence.

3.3 Temps de convergence en fonction de n

La Figure 6 (à gauche) représente l'évolution du temps de convergence en fonction de n, dans le cas $\varepsilon = 0.01$. Le temps de convergence semble exponentiel en n. On a tracé sur la même figure (à droite) la variation, en fonction de n, de $1/d$ où d est la densité des solutions au problème des n reines (quotient du nombre de solutions par le nombre total de positions).

Le nombre de solutions est une valeur tabulée par recherche exhaustive [1]. On constate que d décroît également exponentiellement, ce qui tend à expliquer la croissance rapide du temps de convergence. L'irrégularité au point $n = 6$ s'explique de même par le fait que la variation du nombre de solutions au problème présente la même irrégularité.

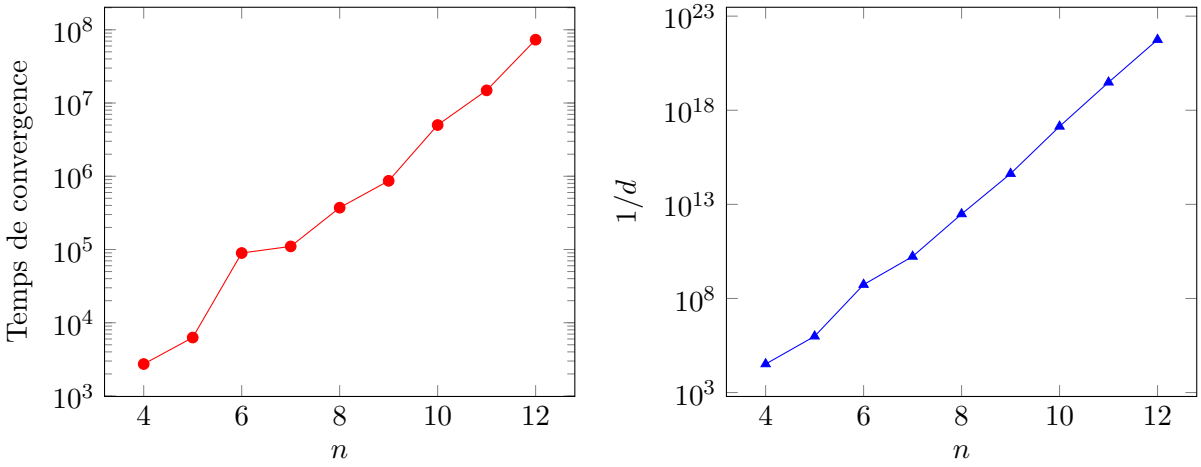

FIGURE 6 – Temps de convergence pour $\varepsilon = 0.01$ à gauche, et inverse de la densité de solutions en fonction de n (échelle logarithmique) à droite.

4 Un autre algorithme

Le modèle précédent ne passe pas très bien à l'échelle : obtenir une solution pour des tailles supérieures à 12 peut s'avérer très long. On va donc envisager un nouveau modèle dans lequel plus d'informations sont prises en compte à chaque étape de calcul. L'objectif est que l'évolution du système soit plus dirigée que précédemment, tout en se restreignant encore à des règles de mise à jour locales. Plutôt que de choisir une paire de cases à chaque pas de temps et ne considérer que les informations contenues dans ces deux cases, on va mettre à jour une case par pas de temps, en utilisant les informations de toutes ses voisines.

On utilise le même système de signaux que dans l'algorithme précédent, avec le même espace d'états : $\mathcal{Q} = \mathcal{P}^8 \times \mathbb{N}$. On pose, pour tout $c \in \mathcal{L}$, $\mathcal{V}_c \subset \mathcal{L}$ l'ensemble des cellules voisines de c : $\mathcal{V}_c = \{c' \in \mathcal{L}, \|(c - c')\|_\infty = 1\}$. À chaque pas de temps, on choisit une case au hasard, on met à jour ses signaux par rapport aux cellules voisines, et une reine quitte éventuellement la case. Contrairement à l'algorithme précédent, comme les signaux d'une seule case sont modifiés à chaque pas de temps, on les met à jour avant d'envisager un déplacement de reine. On met à jour les signaux d'une cellule c par rapport à chacune de ses voisines, de la même façon que dans le modèle précédent : pour toute cellule $c' \in \mathcal{V}_c$, le signal en c provenant de c' apparaît ou reste présent s'il y en a un en c' provenant de la même direction, ou s'il y a une reine en c'. Dans le cas contraire, il disparaît ou n'apparaît pas.

L'idée de la procédure pour déplacer une reine est la suivante : si on met à jour une case c contenant des reines menacées, l'une d'entre elles choisit une case de $\mathcal{V}_c$, et tente de s'y déplacer selon la même procédure que dans l'algorithme précédent. On définit $\omega : \mathcal{L} \to \mathbb{N} \cup \{\infty\}$ qui sert

1. http ://jsomers.com/nqueen_demo/nqueens.html

à quantifier à quel point une cellule est dangereuse, de la façon suivante :

$$\omega(c) = \begin{cases} \infty \text{ si } n_c > 0 \\ \sum_{d \in \mathcal{D}} s_{c,d} \text{ sinon} \end{cases}$$

On aimerait toujours envoyer une reine menacée de c vers une cellule de V_c qui minimise ω, mais il existe des positions pour lesquelles un tel choix déterministe ne permet pas de résoudre le problème. Ainsi, la position de la Figure 7, avec les signaux convenablement propagés (ie. une case porte un signal venant d'une certaine direction si et seulement si il y a effectivement une reine dans cette direction), est une position insoluble. En effet, en partant de cette position, on peut vérifier que les reines en a3 et b3 resteront dans le triangle rouge alors que les autres reines, non menacées, ne bougeront pas.

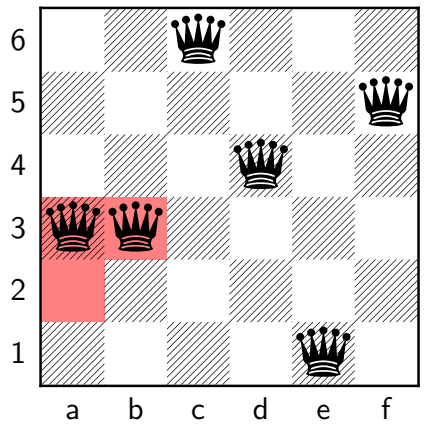

FIGURE 7 – Exemple de position insoluble avec $\eta = 0$.

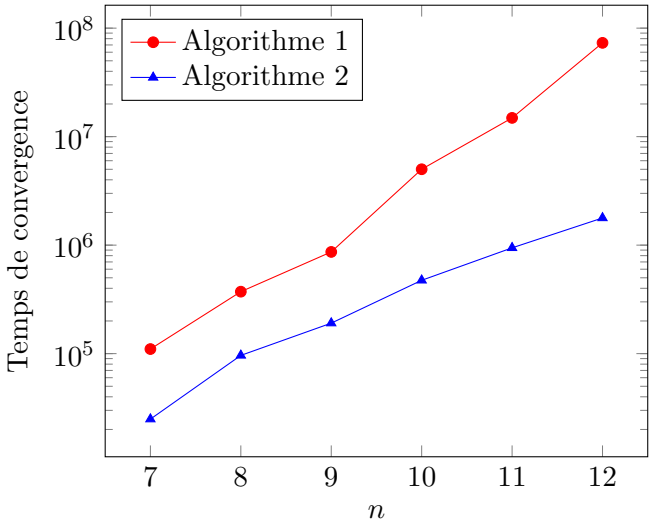

FIGURE 8 – Temps de convergence moyen selon n.

On veut donc que le choix de la destination ne soit pas toujours la case adjacente qui minimise ω. On se donne donc une probabilité $\eta \in [0, 1]$, et on choisit avec une probabilité η une case aléatoire de V_c comme destination, et avec une probabilité $1 - \eta$ une case aléatoire parmi les éléments de V_c qui minimisent ω. Ainsi, la position de la Figure 7 ne pose pas de problème pour $\eta > 0$: une reine va finir par sortir du triangle rouge et une des autres reines se déplacera par la suite. Une fois la destination choisie, la procédure qui décide si le déplacement se fait ou non est exactement la fonction MOVE de l'algorithme précédent.
La Figure 8 montre que ce nouvel algorithme passe mieux à l'échelle que précédemment : la pente de la courbe qui représente le temps de convergence moyen en fonction de n est plus faible pour le nouvel algorithme que pour l'ancien. De fait, il permet de trouver en un temps raisonnable (quelques minutes sur une machine individuelle) des solutions jusqu'à $n = 16$, contre 12 pour le modèle précédent, alors que les temps de calcul sont très similaires pour $n = 8$.

5 Bilan sur le problème des n reines et lien avec la factorisation

On est parvenu à élaborer un automate cellulaire qui résout le problème des n reines. Les performances, en temps de calcul, sont cependant bien inférieures à une simple recherche en profondeur, bien que le caractère purement local des interactions entre cellules permette d'envisager de paralléliser cet algorithme de manière efficace. Par ailleurs, l'utilisation du hasard

permet de conserver des règles locales relativement simples tout en assurant l'accessibilité des solutions. Enfin, on tire de ce problème des enseignements utiles pour la suite : les contraintes à propager qui apparaissent dans le problème de la factorisation peuvent se traduire de manière assez analogue aux signaux transmis par les cellules de ce modèle en utilisant les propriétés de la multiplication binaire.

Multiplication binaire et factorisation

1 Position du problème

...[4] [1] [3]... citations liées à la factorisation ...

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 9. 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 9).

$$M = \begin{matrix} 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{matrix}$$

									$\times$
					1	0	1	1	1
				0	0	0	0	.	0
			1	0	1	1	.	.	1
		1	0	1	1	.	.	.	1
$\sigma(M) =$	1	0	0	0	1	1	1	1	143

FIGURE 9 – La matrice M s'interprète comme la multiplication de 11 (1011 horizontalement) par 13 (1101 verticalement, de haut en bas), et le produit est $\sigma(M) = 143$.

Définition. *Étant donné une matrice M à coefficients dans $\{0,1\}$, on appelle* somme *de M et on note $\sigma(M)$ l'entier obtenu en sommant ses lignes décalées (cf. Figure 9). Si M représente une multiplication, $\sigma(M)$ est par construction le produit.*

L'idée est donc de partir d'une matrice à coefficients dans $\{0,1\}$ dont la somme vaut le nombre qu'on cherche à factoriser, et de lui appliquer des transformations qui préservent la somme comme invariant jusqu'à atteindre une matrice qui représente une multiplication. Ainsi, notre système n'a pas à vérifier que le produit de cette multiplication est l'entier à factoriser, ce qui simplifie considérablement sa conception.

2 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 $n-1 \times \lceil \frac{n}{2} \rceil$ de somme 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 à l'aide d'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 (cf. Figure 10). On vérifie aisément que $\sigma(M) = N$.

$$21 \Rightarrow \overline{10101}^2 \Rightarrow \begin{matrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{matrix} \Rightarrow \begin{array}{cccccccc} & & & & 0 & 1 & 0 & 1 \\ + & & & 1 & 0 & 0 & 0 & . \\ + & 0 & 0 & 0 & 0 & . & . & \\ \hline & & 0 & 1 & 0 & 1 & 0 & 1 \end{array}$$

FIGURE 10 – Exemple de construction de la matrice M de départ pour factoriser $N = 21$.

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 préservent l'invariant de somme. Elles sont représentées sur la Figure 11, et s'appliquent aux coefficients indiqués en gras sous certaines conditions. Les règles **R5** et **R6** ne sont en fait pas nécessaires pour l'accessibilité des solutions : on les introduit par symétrie, pour ne pas privilégier de direction de déplacements des 1.

$$\begin{pmatrix} . & . & \mathbf{1} & . \\ . & . & . & 0 \\ . & . & . & . \end{pmatrix} \underset{\text{R2}}{\overset{\text{R1}}{\rightleftarrows}} \begin{pmatrix} . & . & \mathbf{0} & . \\ . & . & . & 1 \\ . & . & . & . \end{pmatrix} \qquad \begin{array}{cccccc} & & & . & . & \mathbf{1} & . \\ + & & . & . & . & 0 & \\ + & . & . & . & . & & \\ \hline \end{array} \underset{\text{R2}}{\overset{\text{R1}}{\rightleftarrows}} \begin{array}{cccccc} & & & . & . & \mathbf{0} & . \\ + & & . & . & . & 1 & \\ + & . & . & . & . & & \\ \hline \end{array}$$

$$\begin{pmatrix} . & . & 0 & . \\ . & \mathbf{1} & 0 & . \\ . & . & . & . \end{pmatrix} \underset{\text{R4}}{\overset{\text{R3}}{\rightleftarrows}} \begin{pmatrix} . & . & 1 & . \\ . & \mathbf{0} & 1 & . \\ . & . & . & . \end{pmatrix} \qquad \begin{array}{cccccc} & & & . & . & 0 & . \\ + & & . & . & \mathbf{1} & 0 & \\ + & . & . & . & . & & \\ \hline \end{array} \underset{\text{R4}}{\overset{\text{R3}}{\rightleftarrows}} \begin{array}{cccccc} & & & . & . & 1 & . \\ + & & . & . & \mathbf{0} & 1 & \\ + & . & . & . & . & & \\ \hline \end{array}$$

$$\begin{pmatrix} . & \mathbf{1} & 0 & . \\ . & . & . & 0 \\ . & . & . & . \end{pmatrix} \underset{\text{R6}}{\overset{\text{R5}}{\rightleftarrows}} \begin{pmatrix} . & \mathbf{0} & 1 & . \\ . & . & . & 1 \\ . & . & . & . \end{pmatrix} \qquad \begin{array}{cccccc} & & & . & \mathbf{1} & 0 & . \\ + & & . & . & . & 0 & \\ + & . & . & . & . & & \\ \hline \end{array} \underset{\text{R6}}{\overset{\text{R5}}{\rightleftarrows}} \begin{array}{cccccc} & & & . & \mathbf{0} & 1 & . \\ + & & . & . & . & 1 & \\ + & . & . & . & . & & \\ \hline \end{array}$$

FIGURE 11 – Transformations locales préservant l'invariant de somme sur un exemple de 3 lignes et 4 colonnes.

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. 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 12 en donne un exemple. Il est donc nécessaire d'autoriser au moins certains mouvements qui ne sont pas directement utiles en ce sens. Par ailleurs, 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 au 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. Dans tous les cas, on ne s'intéresse pas au caractère sûr ou dangereux de la case de destination.

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

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

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

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

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.

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

FIGURE 12 – Exemple de situation bloquée si on interdit les mouvements ne déplaçant pas les zéros menacés (encadrés) : aucune des règles de la Figure 11 ne s'applique à eux.

3 Atteignabilité de la solution

On cherche ici à montrer que la solution est accessible à partir de toute configuration initiale valide en appliquant uniquement les règles de Γ. On va en fait montrer un résultat un peu plus général : en appliquant les règles de Γ, il est possible de passer d'une matrice quelconque à n'importe quelle autre matrice de même somme. On suppose fixée la taille des matrices considérées, mais on ne la suppose pas nécessairement égale à celle qu'on utilise dans l'algorithme : cette généralisation permet d'adapter la preuve à des améliorations de l'algorithme.

Définition. *Pour toutes matrices A et B, on a $A \sim B$ si et seulement si on peut passer de A à B en appliquant des transformations de Γ.*

Remarquons que $\sim$ est une relation d'équivalence (la symétrie vient du fait que toutes les règles de Γ ont un inverse, donc tous les mouvements sont réversibles).
Pour toute matrice M, on numérote ses diagonales descendantes en commençant à 0 par celle réduite au coin supérieur droit (cf Fig. 13), et on note b_i le nombre de coefficients égaux à 1 sur la diagonale i, de sorte que $\sigma(M) = \sum_i b_i 2^i$. On constate que deux matrices ayant les mêmes coefficients b_i pour

tout i sont équivalentes pour $\sim$ (en fait, on peut passer de l'une à l'autre en utilisant uniquement les règles **R1** et **R2**). On note de plus r le nombre de diagonales descendantes de la matrice (si la matrice est de taille $n \times p$, alors $r = n + p + 1$). Enfin, pour tout $0 \leq i < r$, on note t_i la taille de la i-ème diagonale et l'on a $0 \leq b_i \leq t_i$.

On va montrer que pour tout N, toute matrice de somme N est équivalente pour $\sim$ à une certaine matrice qui contient un nombre minimal de 1.

Soit donc $N \in \mathbb{N}$, m le nombre minimal de 1 que doit contenir une matrice M de somme N. Soit M une telle matrice. Elle vérifie nécessairement, pour tout $i < r-1$, $b_i \leq 1$ ou $b_{i+1} = t_{i+1}$, sinon on pourrait appliquer les règles **R1** et **R2** dans les diagonales i et $i+1$, puis une fois la règle **R4** et se ramener à une matrice avec strictement moins de 1. Par conséquent, si $b_i > 1$, on a par récurrence que pour tout $j > i$, $b_j = t_j$. On suppose que tous les coefficients de M ne sont pas égaux à 1 (sinon, M telle que $\sigma(M) = N$ est unique et on a fini). Il existe alors $k \in [\![0, r-1]\!]$ tel que $b_k < t_k$, pour tout $i < k$, $b_i \leq 1$ et pour tout $i > k$, $b_i = t_i$ (k est le plus grand i tel que $b_i < t_i$). Soit maintenant M' une autre matrice telle que $\sigma(M') = N$, et minimale pour le nombre de coefficients égaux à 1 dans sa classe d'équivalence pour $\sim$. Notons, pour tout i, b'_i le nombre de coefficients égaux à 1 sur la i-ème diagonale de M'. Il existe également k' tel que $b'_{k'} < t_{k'}$, pour tout $i < k'$, $b'_i \leq 1$ et pour tout $i > k'$, $b'_i = t_i$. Si $k' < k$, alors pour tout $i > k$, $b_i = b'_i$ donc :

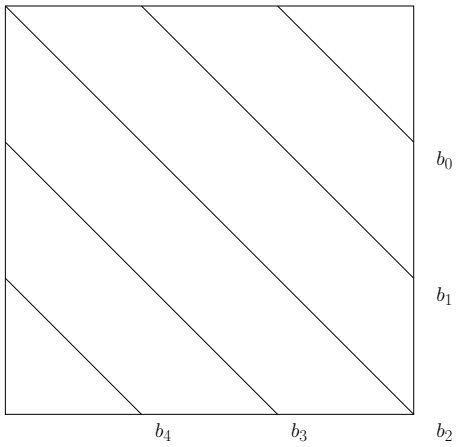

FIGURE 13 – Signification des coefficients b_i.

$$\sum_{i=0}^{r-1} b_i 2^i = N = \sum_{i=0}^{r-1} b'_i 2^i \quad \text{, ce qui implique} \quad \sum_{i=0}^{k} b_i 2^i = \sum_{i=0}^{k} b'_i 2^i$$

$$\text{et} \quad b_k 2^k + \underbrace{\sum_{i=0}^{k-1} b_i 2^i}_{< 2^k \text{ car } b_i \leq 1 \text{ pour } i < k} = \underbrace{\sum_{i=k'+1}^{k} t_i 2^i + b_{k'} 2^{k'} + \sum_{i=0}^{k'-1} b_i 2^i}_{\geq t_k 2^k}$$

ce qui nous amène à une contradiction, car $b_k < t_k$. D'où $k \leq k'$, et par symétrie $k = k'$. Ainsi, $\forall i < k$, $b_i = b'_i$ par unicité de la décomposition en base 2 du reste de N modulo 2^k, et $\forall i > k$, $b_i = b'_i = t_i$, donc on a aussi $b_k = b'_k$ (car $\sum_{i=0}^{r-1} b_i 2^i = \sum_{i=0}^{r-1} b'_i 2^i = N$). D'où $M' \sim M$.

Soit donc A telle que $\sigma(A) = N$, et A' la matrice avec le moins de 1 dans la classe d'équivalence de A. Clairement $A \sim A'$, or on a vu $A' \sim M$ par minimalité de A' dans sa classe d'équivalence. D'où par transitivité, $A \sim M$. Enfin, toujours par transitivité, si B est une autre matrice de somme N, alors $B \sim M$ comme précédemment, donc $A \sim B$.

Toutes les matrices de même somme sont donc équivalentes pour $\sim$, donc on peut passer de l'une à l'autre en appliquant des règles de Γ. En particulier, toute solution est accessible à partir de toute configuration initiale valide, c'est-à-dire de même somme.

4 Performances et observations

Ajouter un bref contexte expérimental (type ordi, OS, langage développement)

Les performances varient grandement selon l'entier à factoriser : sa décomposition en facteurs premiers, et plus généralement la distribution de 1 dans la ou les solutions semblent être des facteurs déterminants. Ainsi, les nombres les plus faciles semblent être ceux qui ont le plus de diviseurs, ce qui s'explique par le fait que plus de solutions existent pour de tels nombres. Si on ne s'intéresse qu'aux nombres qui sont le produit de deux nombres premiers, les nombres pour lesquelles les matrices solutions du problème comportent peu de 1 semblent les plus difficiles : l'algorithme trouve en 10^6 itérations en moyenne un diviseur de 493 ($= 17 \times 29$), mais pour factoriser 362 ($= 2 \times 181$), il lui faut 10^9 étapes ! 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.

5 Quelques améliorations

5.1 Fixer la taille des diviseurs recherchés

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 $4n-2$ 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 $4n - 2$ 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 (condition nécessairement vérifiée par la matrice dont les dimensions sont exactement les tailles de deux

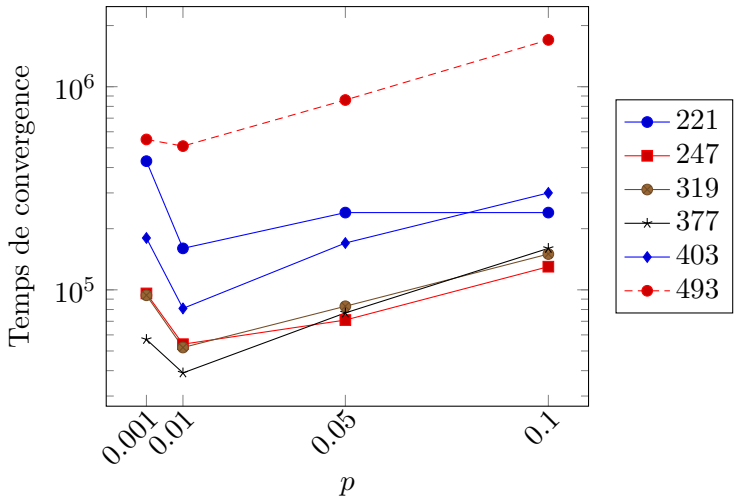

FIGURE 14 – Temps de convergence moyen en fonction de N et p (échelle logarithmique).

facteurs du nombre à factoriser).
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 plus rapide car on ne perd plus de temps à cause des 1 situés dans une zone inutile de la matrice.

On donne en Table 1 les temps moyens mis pour atteindre une solution pour certaines valeurs de N par l'algorithme initial et par l'algorithme dans lequel on suppose connue la taille des diviseurs, lancé avec la combinaison de tailles la plus favorable en moyenne. On tient compte de l'ordre des tailles : en effet, échanger les dimensions de la matrice revient à transposer la matrice solution, mais ce n'est pas le cas de la configuration initiale. Ainsi, il peut y avoir une différence de performance significative : trouver les diviseurs de 901 ($= 17 \times 53$) est en moyenne cinq fois plus rapide sur une matrice de taille 6×5 que sur une matrice de taille 5×6. On constate que la différence de performance avec l'algorithme initial semble s'accentuer quand N croît : cette amélioration semble permettre à l'algorithme de passer beaucoup mieux à l'échelle.

N	algo initial	taille des diviseurs connue	et informations purement locales	et déplacements globaux
221	1×10^6	8×10^4	6×10^4	4×10^4
403	8×10^4	1×10^4	8×10^3	6×10^3
493	5×10^5	1×10^5	1×10^5	1×10^5
635	$> 10^9$	3×10^5	4×10^5	2×10^5
893	3×10^6	2×10^5	3×10^5	2×10^5
899	1×10^6	8×10^3	9×10^3	5×10^3
901	2×10^7	3×10^6	3×10^6	3×10^7
923	$> 10^9$	3×10^5	2×10^5	1×10^5

TABLE 1 – Temps moyen mis pour atteindre une solution (en nombre d'étapes).

5.2 N'utiliser que des informations locales

Dans l'algorithme ci-dessus, on utilise des informations globales sur le système lorsqu'on décide si on applique une transformation : en effet, on décide si une case est sûre ou dangereuse en comptant les 1 sur la ligne et la diagonale concernées. On peut en fait ajuster l'algorithme pour se restreindre à utiliser uniquement des informations locales, en mettant en place un système de signaux analogue à celui utilisé pour le problème des n reines.
On procède de la façon suivante : les cellules de la matrice, en plus de contenir un 0 ou un 1, peuvent contenir des signaux en provenance des quatre directions N, S, E et W. On souhaite qu'un signal en provenance d'une certaine direction soit présent sur une cellule s'il existe un coefficient égal à 1 dans cette direction. Ainsi, une case est dite *sûre* si elle ne contient pas de signal en provenance du nord ni du sud, ou bien si elle ne contient pas de signal en provenance de l'ouest ni de l'est. Dans le cas contraire, elle est dite *dangereuse*. On a ainsi traduit l'ancienne définition d'une cellule sûre (pas de 1 sur la même colonne, ou bien pas de 1 sur la même ligne) en terme de signaux.
À chaque pas de temps, on choisit une cellule c à mettre à jour et on commence par mettre à jour ses signaux comme pour le problème des n reines : pour chaque direction d, si c' est la cellule voisine de c dans la direction d, le signal en c en provenance de la direction d devient présent si c' contient un 1, et prend le même état (présent ou absent) que celui de c' en direction de d sinon. Après avoir mis les signaux de c à jour, on applique l'algorithme précédent pour décider si on applique une transformation de Γ, en utilisant la nouvelle définition de case sûre/dangereuse.
Comme on pouvait s'y attendre, on constate une légère déterioration des performances lorsqu'on se restreint ainsi à des informations locales : chaque pas de calcul est plus long à réaliser (on doit à la fois appliquer des transformations et mettre à jour des signaux), mais on en fait généralement un peu moins. Dans l'ensemble, le temps de calcul est typiquement doublé. Cette déterioration est néanmoins assez raisonnable, ce qui est prometteur dans le sens où se restreindre à des informations locales peut permettre de paralléliser le système en effectuant plusieurs mises à jour simultanées, sans conflits à condition que les zones affectées par les différentes mises à jour soient disjointes. Ceci était moins efficace avec l'algorithme initial du fait de la nécessité de garder une trace du nombre total de 1 sur chaque ligne et diagonale, ce qui impose de centraliser les mises à jour sous peine d'en effectuer certaines avec des informations erronées.

5.3 Permettre des échanges à plus longue distance

Les transformations préservant l'invariant de somme introduites plus haut sont très localisées : on peut envisager de remplacer les règles **R1** et **R2** par une règle plus générale, qui consiste à intervertir deux cellules situées sur la même diagonale descendante. De même, on peut remplacer les règles **R3** et **R5** par une règle qui consiste à échanger un 1 d'une diagonale contre deux sur la diagonale à sa droite, et les règles **R4** et **R6** par une règle qui consiste à échanger un 0 d'une diagonale contre deux sur la diagonale à sa droite. Les transformations du système sont ainsi plus équitables au sens où on peut maintenant atteindre, en appliquant une seule règle, des positions qu'on ne pouvait précédemment atteindre que par une séquence précise de transformations locales.
L'effet de ce changement sur les performances est mitigé : dans la plupart des cas, on constate une amélioration de l'ordre de 40%, mais dans certains cas, le temps de calcul est au contraire beaucoup plus grand (ainsi, il est dix fois plus long pour $N = 901$). Par ailleurs, il devient bien plus difficile de paralléliser les calculs comme on l'a envisagé dans la section précédente : utiliser un automate cellulaire perd de son intérêt.

Modifier la dernière phrase car au contraire, c'est l'extension des voisinages, plus éloignée des ACs, qui perd de son intérêt

Bilan sur le problème de la factorisation

...

Faire ressortir les points suivants :
- Formulation cellulaire du problème
- démonstration de l'atteignabilité de la solution
- étude expérimentale du comportement

Conclusion

...

- Résumé du travail effectué
- Résultats encourageants sur la faisabilité du concept
- Perspectives...

Références

[1] Jacques M. Bahi and Sylvain Contassot-Vivier. Basins of attraction in fully asynchronous discrete-time discrete-state dynamic networks. *IEEE Transactions on Neural Networks*, 17(2) :397–408, 2006.

[2] Vincent Chevrier and Nazim Fatès. Multi-agent Systems as Discrete Dynamical Systems : Influences and Reactions as a Modelling Principle. Rapport de recherche, 2008.

[3] Nazim Fatès. Stochastic Cellular Automata Solutions to the Density Classification Problem - When Randomness Helps Computing. *Theory Comput. Syst.*, 53(2) :223–242, 2013.

[4] Carl Pomerance. A tale of two sieves. *Notices Amer. Math. Soc*, 43 :1473–1485, 1996.