au moins une bonne configuration pour chaque entier $n \geqslant 4$.

Pour trouver une solution pour n quelconque, l'idée la plus simple, proposée dès 1874 par Emil Pauls, est de disposer les reines régulièrement selon un double parcours rectiligne de cavalier, en s'arrangeant pour qu'il y ait une reine sur chaque ligne et une sur chaque colonne.

La méthode est parfaite *(voir échiquiers 1 et 2 de l'encadré page 79)* pour tout entier n multiple de 6 (c'est-à-dire $n = 6m$ où m est un entier), et pour tout n supérieur de 4 unités à un multiple de 6 (c'est-à-dire $n = 6m + 4$). Pour les nombres pairs supérieurs de 2 unités à un multiple de 6 (soit $n = 6m + 2$), la méthode ne convient pas à cause des diagonales *(voir échiquier 3)*. Un autre schéma un peu plus compliqué permet de s'en tirer *(voir échiquiers 4 et 5)*. On a donc une solution pour tout entier pair supérieur ou égal à 4.

Pour les nombres impairs, soit $n = 2m + 1$, on prend une solution pour le nombre pair inférieur d'une unité, soit $2m$, on la place dans le coin en bas à gauche de l'échiquier de côté $2m + 1$, et l'on ajoute une reine dans le coin en haut à droite *(voir échiquiers 6 et 7)*.

Le problème *(i)* de trouver une solution pour tout entier $n \geqslant 4$ est donc résolu… sans avoir utilisé d'ordinateur. Disposer d'une solution, c'est très bien, mais les connaître toutes ou réussir à les compter, ce serait encore mieux. À moins d'une découverte mathématique, l'ordinateur semble cette fois indispensable.

Des ruses pour chercher toutes les solutions

Nous allons envisager trois façons de plus en plus « intelligentes » de rechercher toutes les solutions pour une valeur donnée de n. Cette progression de la puissance quand on améliore les méthodes illustre l'idée qu'en informatique, un programme peut être juste et fonctionner parfaitement pour les petites valeurs d'un paramètre, et pourtant être inefficace car conduisant à des temps de calcul irréalistes (des années, des siècles ou plus !) quand le paramètre augmente.

On commence par écrire un sous-programme qui, quand on lui donne une configuration quelconque de n reines, indique si oui ou non elle est satisfaisante. Un tel sous-programme est facile à concevoir, car deux dames en positions (a, b) et (a', b') sur l'échiquier de côté n (où a, a', b, b' varient de 1 à n) sont en prise l'une avec l'autre si et seulement si $a = a'$ (elles sont sur la même ligne) ou $b = b'$ (même colonne) ou $a - b = a' - b'$ (même première diagonale) ou $a + b = a' + b'$ (même seconde diagonale).

La première idée pour calculer toutes les solutions pour n donné consiste à placer n reines de toutes les façons possibles et pour chaque configuration, d'utiliser le sous-programme qui indiquera si la configuration est solution ou non. Il y a n^2 choix possibles pour la première reine, n^2 choix pour la deuxième, etc., jusqu'à la n-ième. Le nombre de configurations à tester est donc exactement $(n^2)^n = n^{2n}$, ce qui, pour $n = 8$, fait environ $2{,}815 \times 10^{14}$.

La deuxième idée consiste à remarquer que toute solution s'obtiendra en plaçant une reine par colonne, et qu'on peut même s'arranger pour que les reines placées ne soient pas sur les lignes déjà occupées au moment où on les place. Il y a ainsi n façons de placer la première reine dans la première colonne, $n - 1$ façons de placer la deuxième reine sur la deuxième colonne, $n - 2$ façons de placer la troisième reine sur la troisième colonne, etc.

Cela conduit donc à tester en tout $n!$ configurations de n reines. On les teste une à une avec le sous-programme décrit plus haut, qui peut d'ailleurs être simplifié puisque, par construction, les configurations qu'on lui soumet maintenant sont correctes pour les contraintes de lignes et de colonnes. Le gain entre la première méthode et la seconde est considérable. Pour $n = 8$, il n'y a plus que 40 320 configurations à tester, soit environ 7 milliards de fois moins qu'avec la première méthode !

On peut encore faire mieux. La troisième méthode consiste à remarquer que lorsqu'on place les reines une à une comme dans la deuxième méthode, on peut s'apercevoir rapidement qu'on est en train de construire une configuration qui échouera. Si par exemple on a placé une reine en (1, 1), puis la seconde en (2, 2), elles sont sur la même diagonale : il faut donc cesser de construire cette configuration qui ne donnera jamais de solution, quelles que soient les reines qu'on y ajoutera. Découvrir le plus tôt possible deux reines en prise dans l'énumération ordonnée de toutes les possibilités et remettre en cause le choix qui y a conduit est ce qu'on nomme la méthode de *backtracking*, ou retour arrière. Elle est d'usage courant dans de nombreux problèmes combinatoires.

Le gain obtenu par rapport à la seconde méthode est encore très intéressant et augmente avec n. Si l'on compte comme un essai une position où soit on a réussi à trouver une solution, soit on a découvert qu'il sera impossible d'aller plus loin, alors on trouve les 92 solutions pour $n = 8$ en 13 756 essais. Cela correspond à trois fois moins d'essais qu'avec la deuxième méthode. Pour $n = 12$, on n'effectue que 9 261 880 essais au lieu de $12 ! = 479\,001\,600$ avec la deuxième méthode ; on a donc gagné un facteur 50. Notons que la seule façon de connaître le nombre d'essais par la troisième méthode est de l'appliquer complètement, car on ne connaît aucune formule simple donnant ce nombre.

Le record : 26 reines

Il se trouve cependant que pour battre les records, il faut non seulement utiliser une bonne méthode, donc une variante du *backtracking*, mais aussi :

– la programmer très soigneusement ;

– faire exécuter en parallèle les calculs (ce qui n'est pas difficile ici car la méthode s'y prête bien) ;

– fabriquer et utiliser des puces spécialisées qui iront plus vite que les processeurs généraux pour l'exécution des calculs.

C'est ainsi que fonctionne le système informatique aujourd'hui détenteur du record pour $n = 26$. Ce record est détenu par l'université technique de Dresde (projet Queens@TUD), en Allemagne. Les chercheurs ont conçu une méthode massivement parallèle utilisant des circuits FPGA (*field-programmable gate array*) adaptés à ce