Théorie des ensembles appliquée au sudoku et algorithmique associée

Marie-Pierre Falissard, professeure de mathématiques à Pully-Lausanne (collège Champittet)

1. Description ensembliste du sudoku

Une grille de sudoku peut se caractériser par 3 types d'ensembles, qui sont des données de base propres à chaque variante du jeu de sudoku :

a. Un ensemble de symboles S, par exemple le plus courant est S = {1, 2, 9}. Ces symboles seront utilisés pour remplir la grille. On trouve parfois des ensembles plus petits (à 4 symboles) ou plus grands (à 12 ou 16 symboles). On y adjoindra par commodité un symbole particulier : 0, qui correspondra à une case vide. On notera alors : S' = S ∪ {0}.

b. Un ensemble ordonné C de cases c_i. On peut désigner chaque case par un numéro, qui ira ainsi de 1 à 81 pour le sudoku usuel : C = {1, 2, 81}. On utilise aussi une notation positionnelle, par exemple L_iC_j désigne la case intersection de la ligne *i* et de la colonne *j* ; on aurait alors C = {L_1C_1, L_1C_2, , L_9C_8, L_9C_9} pour le sudoku classique à 81 cases, décrit ligne à ligne, de gauche à droite et de haut en bas.

c. Un certain nombre d'ensembles E_j donnés, qui sont des sous-ensembles particuliers de C ("régions"). Chacun de ces ensembles, *a priori* quelconque, est caractérisé par la propriété suivante :

$$\text{card}(E_j) = \text{card}(S),$$

car chacun est destiné à contenir la totalité des symboles de S : il faut donc qu'il y ait autant de cases dans cet ensemble que de symboles possibles.

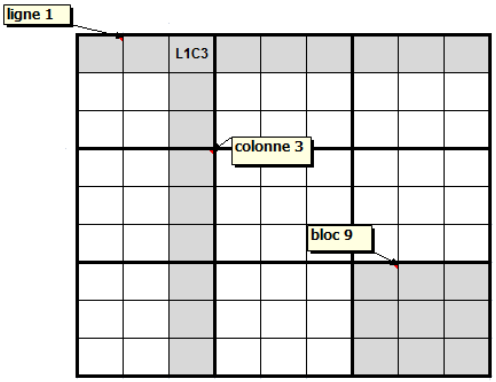
Par exemple, pour des sudokus 9 x 9 :

L_1= {L_1C_1, L_1C_2, , L_1C_9} (première ligne)

C_3= {L_1C_3, L_1C_3, , L_9C_3} (troisième colonne)

B_1= {L_1C_1, L_1C_2, L_1C_3, L_2C_1 , L_3C_3} (premier bloc)

D_1= {L_1C_1, L_2C_2, , L_9C_9} (diagonale descendante)



Le but du jeu consiste à « remplir les cases » en se conformant aux règles du sudoku, c'est-à-dire à définir complètement une application f de C vers S' qui soit une surjection de C vers S vérifiant les propriétés suivantes :

propriété	commentaire
f(C) ≠ {0}	Avant la résolution, certaines cases sont déjà remplies : pour certains c de C, f(c) ≠ 0, pour tous les autres, f(c) = 0 (case vide).
∀ *j*, f(E_j) = S	Après la résolution, il y a non-répétition des symboles dans chaque région (ligne, colonne, bloc, diagonale). La fonction f est donc une bijection de chaque E_j vers S.

2. Algorithme de résolution de grille par force brute

L'algorithme cherche à compléter une grille préalablement initialisée en remplissant chaque case vide l'une après l'autre. Les valeurs initiales sont évidemment supposées constituer une grille valide (pour s'en assurer, on peut faire précéder la routine de résolution d'une étape de vérification qui invoquerait pour chaque case non vide la routine **Validation** indiquée ci-dessous).

La routine de résolution principale peut être de la forme suivante :

Résolution		
En entrée : S ensemble de symboles, C ensemble de cases, (E_j) régions de C vérifiant card(E_j) = card(S).		
	instruction	commentaire
1	Pour toute case c_i de C	Passer en revue successivement chaque case c_i de la grille
2	Si f(c_i) = 0, alors f(c_i) ←**Remplissage**(c_i)	Si la case est vide, chercher à la remplir
3	Si f(c_i) = 0, alors afficher "Impossible"	Si l'on ne peut remplir une case, la grille est impossible

Pour remplir chaque case, la routine **Remplissage** ci-dessous est invoquée. Cette routine a la particularité d'être ***récursive*** (elle s'invoque elle-même en ligne 5). Cette récursivité permet de remplir progressivement la grille jusqu'à ce qu'une impossibilité oblige à faire marche arrière (instruction "f(c_i) ← 0" en ligne 5) d'un ou plusieurs niveaux pour tester d'autres valeurs possibles ("*backtracking*"). La ligne 1 (remplissage terminé) donne une condition d'arrêt pour éviter une boucle sans fin.

Quand la dernière case vide est remplie, la grille est constituée. Si l'une des cases ne peut être remplie après avoir essayé toutes les valeurs permises et qu'il est impossible de revenir en arrière, la grille est déclarée impossible.

Remplissage		
En entrée : case c_i (case vide de C à remplir)		
En sortie : s (valeur proposée f(c_i) pour remplir la case c_i) ; 0 si la case ne peut être remplie		
	instruction	commentaire
1	Si *i* > card(C) alors retour (on renvoie une valeur quelconque non nulle)	Si l'on a atteint la dernière case de la grille, la recherche est terminée.
2	Pour tout élément s de S	s désigne un symbole candidat pour être affecté à la case c_i.
3	Si **Validation**(c_i , s) = OK, alors :	Si la valeur s placée dans la case c_i respecte les règles du sudoku,
4	f(c_i) ← s	elle est candidate pour être affectée à cette case,
5	si **Remplissage**(**casevidesuivante**(c_i))=0 alors f(c_i) ← 0	sauf s'il est impossible de remplir alors la case vide suivante.
6	Si f(c_i) = 0, tester le symbole s suivant, sinon retour de f(c_i)	Si la valeur s ne convient pas, essayer une valeur suivante, sinon garder cette valeur pour la case.
7	Si tous les s de S ont été testés et f(c_i) = 0, on renvoie 0	Impossible d'affecter une valeur à la case c_i (toutes les valeurs possibles de S ont été essayées) .

La ligne 2 n'indique pas de quelle façon la valeur s est choisie dans S : en première approche, on peut procéder séquentiellement, dans l'ordre des symboles (par exemple 1, 2, ...9).

La routine **Validation** est destinée à vérifier que la valeur candidate ne contrevient pas aux règles du sudoku :

Validation		
En entrée : case c (case vide de C à remplir), valeur s		
En sortie : OK si la valeur s convient pour la case c ; KO sinon		
	instruction	commentaire
1	Pour tout ensemble E_j	Passer en revue chaque ensemble (ligne, colonne, bloc, diagonale...)
2	Si c ∈ E_j et s ∈ f(E_j – {c}), alors retourner KO	Si une autre case de l'ensemble E_j a la même valeur s, cette valeur ne convient pas pour c
3	retourner OK	Aucune case de l'ensemble E_j n'a la valeur s

La routine **casevidesuivante** donne la case vide qui suit immédiatement une case donnée :

casevidesuivante
En entrée : case c_i (case en cours)
En sortie : c_j : la case vide qui suit la case c_i (dans l'ordre adopté pour C), sinon une valeur quelconque supérieure à card(C)

En pratique, les données de base (S, C et les E_j), caractéristiques du sudoku, ne sont pas définies dans le programme, mais sont "externalisées" dans un module séparé, qui décrit la variante de sudoku à laquelle on a affaire. Cela permet d'utiliser le même algorithme pour différentes variantes du sudoku (sudoku à symboles différents, à grille plus grande ou "multi-grilles", à régions différentes dans la grille). Ci-dessous, générés par l'algorithme, un sudoku à deux diagonales et un sudoku à 13 blocs (4 blocs en grisé auxquels s'ajoutent les 9 blocs du sudoku courant) ; ces deux grilles inédites[1] sont considérées l'une comme facile, car résoluble par des stratégies élémentaires, l'autre comme de niveau moyen, car résoluble par des stratégies de difficulté moyenne (détection de "paires nues"[2]), moins abordables cependant pour le débutant.

	1	2	3	4	5	6	7	8	9
1	9								4
2			7				8		
3		4			6			5	
4				2		6			
5			8				9		
6				8		4			
7		5			2			3	
8			1				5		
9	3								7

	1	2	3	4	5	6	7	8	9
1									
2			9				1		
3		1	2	4		5	6	7	
4			6				2		
5					6				
6			4				7		
7		6	8	5		1	4	9	
8			1				8		
9									

3. Unicité de la solution

L'algorithme de résolution ne s'assure pas de l'unicité de la solution, il s'assure seulement de son existence.

Pour vérifier l'unicité, on va chercher une deuxième solution et vérifier si elle coïncide avec la première. On effectue cela en réexécutant l'algorithme de résolution en modifiant la ligne 2 de la routine de remplissage pour "choisir différemment" la valeur à tester. Une possibilité, pour trouver une éventuelle "deuxième solution" aussi différente de la première que possible, consiste à tester pour une case donnée le symbole qui suit immédiatement la valeur trouvée pour cette case lors du calcul de la première solution (si "2" était la première solution pour cette case, on va tester, dans l'ordre : 3, 4, ..., 9, 1 et 2 au second passage).

4. Algorithme de génération de grille

La génération d'une grille consiste à remplir une grille vide sans autre règle que le respect des contraintes du sudoku. La génération d'une grille peut être considérée comme un cas particulier de résolution : c'est en fait la résolution d'une grille initialement vide (f(C) = {0}).

[1] Solutions détaillées sur http://tinyurl.com/2sudokus

[2] Par exemple, on constatera, au cours de la résolution du sudoku de droite, que $f(\{L_6C_4, L_6C_6\}) = \{2,3\}$, ce qui détermine une "paire nue" {2,3}. On en déduira alors que $f(L_6C_2) = 9$, seule valeur possible pour la case L_6C_2.

Mais si l'on applique l'algorithme de résolution ci-dessus sans changement, on va générer systématiquement la même grille triviale (ci-contre), remplie ligne à ligne de gauche à droite et de haut en bas, où l'on met dans chaque case la plus petite valeur possible.

1	2	3	4	5	6	7	8	9
4	5	6	7	8	9	1	2	3
7	8	9	1	2	3	4	5	6
2	1	4	3	6	5	8	9	7
3	6	5	8	9	7	2	1	4
8	9	7	2	1	4	3	6	5
5	3	1	6	4	2	9	7	8
6	4	2	9	7	8	5	3	1
9	7	8	5	3	1	6	4	2

Pour que la grille soit différente à chaque génération, le choix de s dans S doit être aléatoire au lieu d'être séquentiel. Cela se réalise aisément par emploi, dans la ligne 2 de la routine de remplissage, d'une fonction aléatoire (choix d'un entier au hasard entre 1 et 9 pour le sudoku classique). Il faut seulement veiller à ce que la boucle de test (lignes 2 à 6) ne réutilise pas plusieurs fois un symbole déjà testé auparavant dans cette même boucle.

Une fois une grille générée (et donc remplie), on supprime des chiffres jusqu'à parvenir à une grille ayant un certain nombre de cases remplies (au moins 17 pour le sudoku courant), les autres cases étant vides. On s'assure enfin de l'unicité de la solution.

On peut aussi, par souci d'esthétique, créer une grille ayant un motif prédéfini (ainsi les deux exemples donnés plus haut ont un pré-remplissage de valeurs respectant une symétrie centrale). Il suffit, soit de choisir parmi un grand nombre de grilles générées aléatoirement celles qui répondent au motif concerné, soit de construire la grille à partir de cette contrainte initiale.

5. Algorithmes de résolution "intelligente"

L'algorithme de résolution par force brute vient très rapidement à bout de toutes les grilles valides, mais le joueur "humain" ne procède jamais ainsi (sauf quand il a épuisé tous les procédés "humains" à sa disposition). En effet, le joueur utilise différentes stratégies plus ou moins simples, qui reposent sur les propriétés de bijection de f restreinte aux régions E_j :

- certaines sont élémentaires : chercher où placer un symbole donné dans une ligne, une colonne, un bloc... (étant donnés E_j et $s \in S$, chercher $c \in E_j$ tel que f(c)=s) ; vérifier si une case donnée à la croisée de plusieurs régions pourrait être remplie par un seul symbole candidat (étant donné c, chercher s tel que f(c)=s, en considérant tous les E_j qui contiennent c) ;

- d'autres sont plus élaborées : détecter dans certaines régions des "paires" ou des "triplets" de façon à restreindre les valeurs candidates ; ainsi, dans l'exemple ci-contre, on peut éliminer la possibilité d'une valeur 3 pour L_8C_4 et L_9C_4 parce que les cases en L_8C_5 et L_9C_6 ne peuvent contenir que les valeurs 1 et 3 ("exclusion pour cause de paire nue") ;

	1	2	3	4	5	6	7	8	9
1	5	7			9				4
2		2		7		4		9	
3			3	5	6	8	1	2	7
4			7	1	4		9		
5			2	6	8	7	3	4	
6			4		5	9	7		
7	2		5	9	7	6	4		
8		6		3		5	2	7	9
9	7		9	3	2				6

- d'autres sont très élaborées (dans le pire des cas, on se rapproche du procédé de force brute : on élimine une valeur possible en l'affectant virtuellement à une case et en continuant à remplir la grille jusqu'à tomber sur une impossibilité).

Un des rôles du créateur de grille de sudoku est d'évaluer le niveau de la grille proposée selon la difficulté des stratégies à mettre en œuvre pour sa résolution. Cette difficulté, contrairement à ce qu'on pourrait penser, n'est pas liée au nombre de cases initialement remplies, ni à leur disposition dans la grille. Il semble par chance qu'une grille générée de façon aléatoire soit le plus souvent "facile"...

6. Bibliographie

- *Les Maths au Carré - Algorithmes & spéculations diverses*, Marie-Pierre Falissard (Ellipses, 2011)
- *Précis de sudoku*, Narendra Jussien (Lavoisier, 2006)