Solveur et générateur de Sudoku

WODOME Elom Marc Arnold

21 novembre 2011

Résumé

Le but de ce projet est de réaliser un programme permettant la résolution et la génération de grilles de Sudoku de taille variant de 1x1 à 64x64. Le langage de programmation est le langage C.

Table des matières

1 Introduction

1.1 Le jeu de sudoku

Le jeu du Sudoku consiste à compléter une grille carrée divisée en N régions de N cases, en partie remplie avec des chiffres, de façon à ce que dans chaque ligne, chaque colonne et chaque région les chiffres de 1 à N apparaissent une et une seule fois. La grille standard que l'on connait tous est le 9x9. Quelques cellules contiennent des chiffres, dits « dévoilés ». Le but est de remplir les cellules vides, un chiffre dans chacune, de façon à ce que chaque rangée, chaque colonne et chaque région soient composées d'un seul chiffre allant de 1 à 9. Généralement on utilise les chiffres et les lettres pour représenter les différentes possibilités des cellules.Ainsi, on a seulement le chiffre 1 pour une grille de taille 1, les chiffres de 1 à 4 dans le cas d'une grille de taille 4, les chiffres et les lettres de 1 à 9 et de A à G, pour une grille de taille 16, de 1 à 9 et de A à P, pour un sudoku de taille 25 etc. Voici une exemple de grille 9x9 et sa solution.

grille

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

solution

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

1.2 Les heuristiques

Une heuristique est un algorithme qui fournit rapidement une solution réalisable, pas nécessairement optimale, pour un problème d'optimisation. Une heuristique, ou méthode approximative, est donc le contraire d'un algorithme exact qui trouve une solution optimale pour un problème donné. Nous allons présenter ici quelques heuristiques que nous avons utilisé dans le programme.

Lone number ou Naked Single

Cette heuristique est l'un des plus simples et des plus utilisées pour la résolution de Sudoku. Cette technique s'appuie sur le raisonnement suivant : toutes les cases devant être remplies, lorsque l'une d'elles ne peut plus contenir qu'un seul chiffre, on peut en déduire que ce chiffre ira nécessairement dans cette case. Dans l'exemple ci-dessous le chiffre 3 est le seul candidat à la ligne 1 colonne 1 dans le premier bloc. On doit donc éliminer les 3 sur la première ligne et éventuellement sur la première colonne.

FIGURE 1.1 – Lone Number

Cross-hatching

L'heuristique du cross-hatching consiste à éliminer dans une sous-grille la couleur des cellules déjà résolues dans la sous-grille (colonne,ligne ou région) ainsi cela permet de réduire les possibilités des cases. Dans l'exemple suivant, le chiffre 5 doit être enlevé de toutes les sous-grilles auquel il appartient.

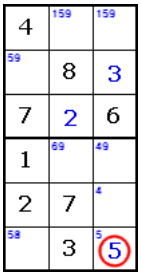

FIGURE 1.2 – Cross-hatching

Naked Subset ou groupes isolés

Si N candidats se retrouvent seuls dans N cases, il est possible de savoir qu'ils finiront bien dans ces N cases. Même si les N candidats ne sont pas présents dans les N cases ! Et donc supprimer ces candidats des autres cases

FIGURE 1.3 – Naked Subset

Hidden Subset ou groupes mélangés

Cette méthode ressemble beaucoup à celle expliquée sous "Groupes isolés". Mais son application n'est pas la même. Car elle ne concerne que les cases qui contiennent les candidats, et pas les autres. Si on ne retrouve N candidats, que dans N cases, on est certain qu'ils vont terminer dans ces N cases. Il est donc possible de supprimer les autres candidats de ces N cases.

FIGURE 1.4 – Hidden Subset

Implémentation des heuristiques

Parlons d'abord ce qu'on appelle "ensemble préemptifs"
Un ensemble préemptif est compose des possibilités qu'on peut avoir pour une case donnée de la grille : 1 à 9 pour les grilles de sudoku par exemple (pour des sudokus de taille 9).
Ces éléments représentent les couleurs qui sont encore possibles dans une cellule.
Pour pouvoir les implémenter, on travaille avec des nombres binaires et on a créé le type pset qui au fait codé sur 64 bits. Ainsi chaque couleur a une position donnée. Plusieurs opérations sont possibles sur les ensembles préemptifs comme l'union, l'union exclusive, l'intersection...
Pour implémenter le lone number on considère les couleurs qui sont présentes une et une seule fois et on parcourt la grille en les enlevant.
Le cross-hatching a été facile à implémenter. Bref, pour chaque sous-grille on recupère la liste de tous singletons et on parcoure la sous-grille pour les enlever des autres cases.
J'ai aussi implémenté l'heuristique Naked Subset. Pour chaque ensemble N d'une sous-grille, je parcoure les autres grilles et je soustrais l'ensemble N des

grilles tout en comptant le nombre de fois où la soustraction aboutit à une ensemble vide. Si ce nombre de fois est égale à la cardinalité de N, alors j'ai un Naked subset de taille N. Je parcours alors la sous-grille en soustrayant chaque couleur de N.
J'ai voulu aussi faire l'heuristique hidden Subset mais une remarque pertinente est que chaque fois qu'il y a un Naked subset, il y a aussi un hidden subset, donc l'un des deux suffit largement. Il y a d'autres heuristiques plus élaborées pour la résolution de sudoku comme le X-wing mais ces heuristiques n'ont pas été implémentées car elles demandent de parcourir en même temps 2 lignes ou 2 colonnes alors que notre programme parcourt une et une seule sous-grille à la fois.

2 Le solveur de grilles

La résolution des grilles se base d'abord sur les heuristiques. certaines des grilles testées ont été résolues uniquememnt par les heuristiques. Mais des fois les heuristiques elles seules ne suffisent pas pour pouvoir résoudre les grilles. on fait alors appel à une technique qui s'appelle le backtracking que je vais expliquer ci-dessous.

2.1 Backtracking

Il s'agit d'une technique plutôt brutale de résolution de problèmes de nature combinatoire. Elle consiste en une exploration systématique de toutes les configurations possibles d'un espace fini totalement ordonné, donc représentable par un arbre fini mais en procédant de telle sorte que le rejet d'une configuration entraine automatiquement celui d'un ensemble d'autres. Ce n'est rien d'autre que l'approche adoptée pour sortir d'un labyrinthe : avancer aussi loin que possible et revenir sur ses pas quand on est bloqué et essayer alors un autre chemin. Dans cette solution, il n'y a aucune intelligence : on explore de manière syséematique toutes les possibilités. Un joueur de sudoku procédant de la sorte remplit la grille avec un crayon et fait un usage intensif de la gomme pour effacer des tentatives infructueuses et revenir en arrière en explorer de nouvelles. Pour pouvoir aboutir à des grilles résolues ou pas suivant les heuristiques implémentées, on se base sur les heuristiques. Donc plus les heuristiques sont rapides et efficaces plus rapidement on trouve la bonne tentative. Dans notre algorithme du programme on choisit la case ayant le plus petit nombre de possibilités et on choisit la couleur à gauche. On a créé alors une méthode stackpush qui permet de choisir une possibilié et une méthode stackpop qui permet de revenir en arrière. La solveur s'arrête des qu'il a trouvé une solution

2.2 Résultats obtenus

La résolution des grilles en combinant les heuristiques et le backtracking est rapide surtout pour les grilles de petite taille (4x4,9x9,16x16,25x25). A partir de 36 et au délà la résolution peut-être lente. Car il y alors plus de

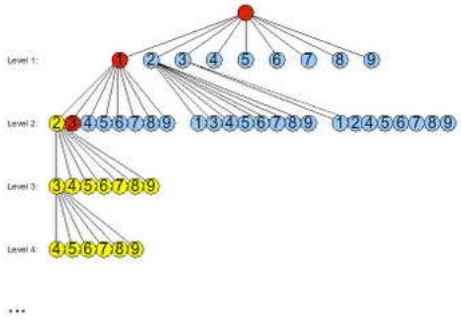

FIGURE 2.1 – Illustation du backtracking

possibilités à explorer Au fait la résolution d'un Sudoku de taille arbitraire est un problème NP-complet.Cela veut dire que l'on peut vérifier facilement qu'une solution est correcte, mais qu'en revanche en trouver une est ardu. Très grossièrement cela veut dire que l'on ne peut pas espérer construire un algorithme pour qu'un ordinateur puisse résoudre ce problème. (Rappelons qu'on parle de grilles de taille arbitraire.)

3 Génération des grilles

3.1 Génération non-stricte

Il y a plusieurs manières de générer les grilles de Sudoku. Notre programme est scindé en 2 étapes : la génération de la grille complète, puis le retrait des chiffres. J'utilise au fait 2 méthodes pour générer les grilles. Pour les grilles de taille inférieure à 25, je créer une permutation des ensembles préemptifs sur une ligne choisie au hasard et je résous la grille avec le solveur. Pour les grilles de taille supérieure,je crée au début une grille trivile et j'utilise des transformations pour générer les autres grilles.Les transformations que j'utilise sont les suivantes : La permutation de 2 lignes ou de 2 colonnes au sein d'un même bloc Une permutation f de l'ensemble préemptif qui transforme aussi une grille solution G en une autre grille solution. Ces transformations sont appliquées un certain nombre de fois (aléatoire).

Génération stricte

Les même principes utilisés pour la génération non-stricte sont utilisés aussi pour la génération stricte sauf que après avoir obtenu la grille solution, on efface les cellules une à une jusqu'à ce que le nombre de solutions possibles dépasse 1. Cela présuppose donc la création d'une méthode qui compte le nombre de solutions. Etant donné que ce nombre peut être très grand suivant la taille de la grille, on s'arrête dès que le nombre de solutions dépasse 1.

Utilisation du programme

Le programme est découpé en 2 parties. Le solveur et toutes les méthodes annexes sont dans sudoku.c, les différents opérations sur les ensembles préemptifs sont regroupés dans preemptiveset.c Pour utiliser le programme, plusieurs options sont disponibles : par défaut on exécute le ficher binaire avec argument le fichier contenant la grille.

- l'option -g permet de générer les grilles avec -s for forcer la génération stricte
- l'option -o redirige l'entrée standard vers un fichier

– l'option -h pour afficher l'aide
– l'option -v pour exécuter le programme en mode verbose (affichage des transformations successives)

4 Tests

Je présente ici les différentes durées de génération des grilles. Les tests sont exécutés sur une machine 32 bits.

Taille de la grille	Génération non-stricte	Génération stricte
1	0,001	0,001
4	0,003	0,004
9	0,012	0,042
16	0,091	0,309
25	0,677	4,3
36	0,008	8,204
49	0,018	1m 16
64	0,061	54.182s

4.1 Interprétation

Au fait , on observe que le temps de génération croit suivant la taille de la grille et suivant l'option stricte ou pas mais en réalité pour les grilles de tailles supérieure à 25 le temps de génération n'est pas uniforme car des fois elle peut être très rapide mais des fois elle peut durer.

5 Conclusion

Les moments les plus difficiles de ce projet ont été l'optimisation et la gestion des erreurs mémoires. L'optimisation se situe surtout au niveau des heuristiques. Car elles réduisent considérablement le temps de résolutions lorsqu'elles sont rapides. Pour la gestion de la mémoire, l'outil valgrind m'a beaucoup aidé. J'ai aussi du utiliser les opérateurs binaires de manière assez importante ce qui m'a permis d'en apprendre plus sur leur fonctionnement et des méthodes qui existent pour faire rapidement des opérations sur les nombres binaires.