

Méthodes de résolution du magic square

Dorian Dumez

December 1, 2016

1 Solveur complet

1.1 Principe

Un solveur complet parcourt virtuellement tout l'arbre des possibles du problème, c'est à dire toutes les permutations possibles. Mais en réalité il tente de construire une solution en essayant de déterminer, le plus tôt possible, si une configuration peut, ou non, mener à une solution. Cette méthode permet donc de déterminer si il n'y a pas de solution, ce de manière exacte, et si il y en a une de la trouver. Tout cela modulo un temps de calcul qui peu être prohibitif.

1.2 Représentation mémoire

L'élément de base de la résolution est la variable. Cette structure représente l'ensemble des informations sur une case de la grille. En effet chaque case à un domaine de valeur possible dépendant du carre magique étudié, de sa position et des valeurs déjà fixés. Au départ toutes les variables ont comme domaine $\{1, 2, \dots, k^2\}$, où k est la taille du carre magique. On a aussi besoin de stocker sa valeur si elle été fixé, on remarque alors que cette valeur sert aussi à indiquer si elle n'a pas été fixé, en lui donnant une valeur non autorisé. De plus, pour la gestion du backtrack on utilise un historique des valeurs qui ont été enlevées de son domaine de valeur possible. Enfin, pour des raisons pratiques, une variable connait sa position dans la grille.

La valeur de la variable est codé par un int. Une valeur admissible étant comprise entre 1 et k^2 , on initialise toujours celle ci à 0 pour indiquer que sa valeur n'a pas encore été choisie.

L'ensemble des valeurs possible est stocké dans un ensemble d'entier (`set<int>`). En effet cette structure permet l'insertion et la suppression de valeur en temps logarithmique. De plus, étant donné qu'ils sont codé avec des AVL, cela pourrait aussi nous permettre d'accéder au extremum de ce domaine en temps logarithmique. Mais l'interface de la `stl` ne le permet pas, donc on les recherche de manière classique, par parcours, donc en $O(taille)$.

Enfin l'historique est une pile d'ensemble d'entier (`stack<set<int>`). En effet, on conserve les avantages des ensembles pour l'ajout et la suppression. Et pour l'historique on ne travaille que sur le dernier ensemble l'ajouté (celui correspondant à la dernière variable

fixé), donc on prend tous les avantages de la pile pour ce genre d'opérations.

1.3 Filtrages

Le filtrage consiste à enlever préventivement des valeurs des domaines de valeurs restantes, car l'affectation de ces dernières ne peut mener à une solution.

Le premier filtrage est celui correspondant au allDiff. C'est à dire que lors de l'affectation d'une valeur à une variable on retire cette valeur de tous les autres domaines. C'est d'ailleurs durant ce procédé que l'on rajoute un nouvel ensemble (parfois vide si cette variable ne pouvait prendre cette valeur ou si elle possède déjà une valeur) au sommet de la pile d'historique.

Le second filtrage est le cassage de symétrie. En effet un carre magique de taille k possède de nombreuses solutions. Donc pour restreindre le parcours de l'arbre des possibles on restreint les solutions acceptables, de manière à être sûr qu'il y en ait toujours une. On a donc sur-contraint le problème avec :

- $c[1, 1] > c[1, n]$ pour casser les symétries gauche/droite
- $c[1, n] > c[n, 1]$ pour casser les symétries haut/bas
- $c[1, 1] > c[n, n]$ pour casser les symétries autour de la diagonale

Le filtrage consiste alors à enlever toutes les valeurs ne respectant pas ces conditions. Par exemple on force toutes les valeurs de $c[1, n]$ à être strictement inférieure au maximum des valeurs restantes de $c[1, 1]$ et toutes les valeurs restantes de $c[1, 1]$ à être strictement supérieur au minimum des valeurs restantes de $c[1, n]$.

Le troisième filtrage consiste à enlever les valeurs qui ne permettront pas d'atteindre les objectifs de somme. C'est à dire que l'on calcule un majorant et un minorant des valeurs possibles des variables d'une ligne/colonne/diagonale en fonction du maximum/minimum des valeurs qui leur restent (ou des valeurs déjà fixées) et de la valeur actuelle de la somme sur cette ligne/colonne/diagonale.

1.4 Parcours

Après expérimentation sur minizinc j'ai choisi de fixer les variables avec le plus petit domaine restant en premier, et le tout en commençant par les valeurs médianes.

On note toutes les variables qu'il reste à fixer grâce à un ensemble de pointeurs vers ces variables (set<variable >). Cela permet l'ajout et la suppression d'éléments très rapidement. Mais la valeur des variables, pour la comparaison (la taille de leur domaine restant), étant changeante on est obligé de parcourir tout l'ensemble à chaque fois pour trouver la variable à fixer (par expérimentation ce procédé améliore tout de même les performances).

Pour éviter la lourdeur du calcul de la médiane on se contente de prendre l'élément du domaine restant qui minimise la distance à $k^2/2$. Étant donné que l'on ne peut pas bénéficier de la structure des set pour la recherche d'extremum on parcourt l'ensemble du domaine restant à chaque fois (cela améliore aussi les performances).

2 Solveur incomplet

2.1 Principe

Le principe de mon solveur incomplet est de générer aléatoirement une grille contenant une unique fois tous les nombres de 1 à k^2 puis, par mouvement dans des voisinages, de trouver une solution. Pour se guider on quantifie la qualité de chaque solution grâce à une fonction d'évaluation. Cette dernière calcule la somme des carrés des différences entre la somme des lignes/colonnes/diagonales et le nombre magique : $\frac{k(k^2+1)}{2}$. De plus, par rapport au voisinage choisis, nous ne sommes pas obligé de re-calculer le score en entier à chaque fois, mais juste ce qui a changé. On parcourt alors le voisinage avec une heuristique de descente guidé par la fonction d'évaluation. Si la fonction tombe dans un minimum local elle effectue un restart pour recommencer d'un autre point de départ.

On remarque tout de suite que cette procédure est très aléatoire. Mais pour pouvoir reproduire les expériences le générateur aléatoire est initialisé avec une graine constante au début du programme.

2.2 Représentation mémoire

La grille est alors juste représentée comme un tableau en 2 dimension de taille $k \times k$ d'entier (`vector<vector<int>>`). De plus pour le re-calcul partiel du score la grille contient son score actuel.

2.3 Voisinages

Deux structures de voisinages ont été créées, mais par expérimentation seules ou combinées, la solution consistant à n'utiliser que la deuxième a été retenue. Au départ l'idée était d'utiliser la première en descente puis, une fois dans un minimum local, d'utiliser la deuxième pour en sortir, car le premier plongeait très vite dans les minima locaux.

Le premier voisinage consistait, pour chaque ligne et colonne, à tester toutes les permutations de cette dernière et de sélectionner celle amenant au meilleur score. Premièrement la plus profonde descente est lente mais en plus cette heuristique plongeait remarquablement vite dans des minima locaux sans intérêt. Donc seule elle nécessitait un très grand nombre de restart, et combinée avec la deuxième c'est cette dernière qui faisait pratiquement tout le travail.

La deuxième structure de voisinage est un swap. On l'applique cette fois-ci en descente, c'est à dire que l'on teste toutes les permutations de deux éléments, et si cela améliore le score on garde la permutation. De plus il faut remarquer que l'on n'arrête

pas le parcours dès la première modification concluante pour ne pas se concentrer trop longtemps sur une même zone, ainsi on peut avoir une progression homogène sur toute la grille. Cette structure de voisinage est nettement plus rapide à utiliser que la précédente et tombe moins souvent dans des minima locaux non solution (c'est à dire n'ayant pas une valeur de 0).

En effet pour utiliser tous ces voisinage il faut remarquer qu'une grille est solution si et seulement si son score est de 0. Cela veut alors dire que la somme de toutes les lignes/colonnes/diagonales sont égales au nombre magique. Et étant donné que l'on ne travaille que sur des permutations le allDiff est nécessairement respecté.

2.4 Autres essais

En dehors des structures de voisinage plusieurs autres essais infructueux ont été effectués.

L'initialisation de la grille avec la permutation identité a été utilisée. Et dans le cas d'un minimum local on effectuait plusieurs permutations dans la grille de manière aléatoire pour en sortir. Mais en expérimentation le premier minimum local atteint n'était pratiquement jamais la solution alors que cela arrive avec une initialisation aléatoire. Cette technique a donc été abandonnée.

D'autres fonctions d'évaluation ont été utilisées, la meilleure (les moindres carrés déjà présenté) a bien entendu été conservée. Une fonction considérait une solution comme meilleure si elle respectait plus de contraintes à l'égalité, mais elle produisait trop de scores égaux. Et une autre utilisait la valeur absolue au lieu du carré, mais elle ne pénalisait pas assez les grands écarts.

3 Expérimentations

Il faut noter que l'algorithme complet indique bien l'absence de solution pour le cas du carre magique de taille 2 tandis que l'incomplet ne s'arrête pas.

taille du carre	1	2	3	4	5	6
temps d'exécution	3.8e-05	7e-05	0.000772	0.008885	9.71918	3.11742

Table 1: temps d'exécution de l'algorithme complet (en seconde)

Au delà les temps de calcul deviennent trop grand.

taille du carre	1	3	4	5	6	7	8	9	10
temps d'exécution	3e-06	1.6e-05	8.9e-05	0.0128	0.0134	0.0643	0.0307	0.108	0.274
rapport plus court/long	∞	0.008	0.030	0.004	0.007	0.012	0.007	0.007	0.009
taille du carre	11	12	13	14	15	16	17	18	19
temps d'exécution	0.183	0.325	0.784	3.40	2.19	1.28	1.22	9.53	25.68
rapport plus court/long	0.014	0.024	0.005	0.006	0.029	0.007	0.016	0.011	0.007

Table 2: temps d'exécution de l'algorithme complet (en seconde)

Les expérimentations sont effectués sur 30 exécutions, le temps donné est une moyenne sur ces 30 exécutions. L'algorithme fonctionne bien en moyenne mais il est très inconstant. En effet la plus longue exécution est souvent, au moins, 100 fois supérieure au temps de la plus rapide.