# Méthodes de résolution du magic square

Dorian Dumez

December 1, 2016

## 1 Solveur complet

### 1.1 Principe

Un solveur complet parcours virtuellement tout l'arbre des possible du problème, c'est à dire toutes les permutation possible. Mais en réalité il tente de construire une solution en essayant de déterminer, le pus tôt possible, si une configuration peu, ou non, mener à une solution. Cette méthode permet donc de déterminer si il n'y a pas de solution, ce de manière exacte, ou 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 information sur une case de la grille. en effet chaque case à un domaine de valeur possible dépendant du carre magique étudie, de sa position et des valeurs déjà fixés. Au départ toutes les variables on comme domaine $\{1, 2, ..., k^2\}$, où k est la taille du carre magique. On a aussi besoin de stocker sa valeur si elle été fixé, on remarque alors que cet 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 on été enlevé à son domaine de valeur possible. Enfin, pour des raisons pratiques, une variable connais 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 stocke 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 donne 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, 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 dernière ne peu 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 nombreuse solutions. Donc pour restreindre le parcours de l'arbre de possible on restreins les solutions acceptables, de manière à être sûr qu'il y en ai 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 a enlever touts 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 possible des variables d'une ligne/colonne/diagonale en fonction du maximum/minimum des valeurs qui leurs reste (ou des valeurs déjà fixé) et de la valeur actuelle de la somme sur cette ligne/colonne/diagonale.

## 1.4 Parcours

Après expérimentation sur minizinc j'ai choisis 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 pointeur vers ces variables (set<variable >). Cela permet l'ajout en la suppressions d'éléments très rapidement. Mais la valeur des éléments 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 donne que l'on ne peut pas bénéficier de la structure des set pour la recherche d'extremum on parcours 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 un grille contenant une unique fois tous les nombres de 1 à $k^2$ puis, par mouvement dans des voisinage, de trouver une solution. Pour se guider on quantifie la qualité de chaque solution grâce à une fonction d'évaluation. Ce 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 somme pas obligé de re-calculer le score en entier à chaque fois, mais juste ce qui à changé. On parcours alors les 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 un graine constante au début du programme.

## 2.2 Représentation mémoire

La grille est alors juste représenté 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 contiens son score actuel.

## 2.3 Voisinages

Deux structure de voisinage on été créé, mais par expérimentation seul ou combiner la solution consistant à n'utiliser que le deuxième a été retenue. Au départ l'idée était d'utiliser le première en descente puis, une fois dans un minimum local, d'utiliser le deuxième pour en sortir, car le premier plongeai très vite dans les minima locaux.

Le premier voisinage consistait, pour chaque ligne et colonne, à tester toutes les permutation de cette dernière et de sélectionner celle amenant au meilleur score. Premièrement la plus profonde descente est lente mais en plus cette heuristique plongeai remarquablement vite dans des minima locaux sans intérêt. Donc seule elle nécessitait un très grand nombre de restart, et combiné 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 ce concentrer trop longtemps sur une même zone, ainsi on peut avoir une progressions 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 un valeur de 0).

En effet pour utilise 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. Mais 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 on été effectué.

L'initialisation de la grille avec la permutation identité à été utilisé. Et dans le cas d'un minimum local on effectuai 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 abandonné.

D'autre fonction d'évaluation ont été utilisé, la meilleure (les moindre carré déjà présenté) à bien entendu été conservé. Une fonction considérai une solution comme meilleure si elle respectai plus de contraintes à l'égalité, mais elle produisait trop de score égaux. et une autre utilisait la valeur absolue au lieu du carré, mais elle ne pénalisait pas asse les grands écarts.

## 3 Tests

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.