UQAC
Université du Québec
à Chicoutimi

Blin Sébastien
Drouin Viallard Victor

Université du Quebec à Chicoutimi

Maîtrise professionnelle
Parcours Informatique

TP 2 8INF846

Réalisation d'un solveur de Sudoku

Sous l'encadrement de :

Bouchard Kévin

1 Introduction

Le but de ce TP est de résoudre des sudokus en implémentant les algorithmes vu durant le cours *Problèmes à satisfaction de contraintes*. Notre implémentation utilise les algorithmes AC3 pour la vérification des contraintes et une combinaison des algorithmes Backtracking et MRV pour la résolution en elle-même.

2 Exécution du programme

2.1 Dépendances du projet

Ce devoir a été développé en *C++14* et a été testé sur des environnements *Linux* et *Windows*. Pour compiler le programme sous Unix programme il vous faut :

- *G++* (version 6 recommandée)
- La librairie *SFML*
- *Make*

2.2 Compilation

Il suffit alors d'exécuter la commande *make* dans le dossier *src*. Un binaire nommé *sudokusolver* sera alors présent.

2.3 Configuration et exécution

Avant de lancer le programme, il faut tout d'abord un sudoku à résoudre. Pour ceci, il faut un fichier de la forme :

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

Puis pour éxécuter le programme

```
./sudokusolver sudoku_file 100
```

Note : 100 étant le délai pour rafraîchir la fenêtre.

Pour windows, il suffit d'ouvrir le programme *cmd.exe*, de se rendre dans le dossier où se trouve l'éxécutable, et de taper :

```
cd XXXXXXX\DOSSIERRENDU\bin\
sudokusolver.exe sudoku_file 100
```

Nous proposons la série d'éxécution suivante :

```
sudokusolver.exe sudoku1 100
sudokusolver.exe sudoku2 100
sudokusolver.exe sudoku3 100
sudokusolver.exe sudoku4 100
```

3 Structure du programme

3.1 Structure générale

La structure du programme est simple. Le programme possède une classe *Sudoku* (qui contient un tableau de *Cell*). Une *Cell* est composée d'une valeur et de valeurs possibles. On donne ensuite le Sudoku à un objet *Solver* qui se charge de la résolution. L'affichage est géré dans la classe *Displayer*.

3.2 La résolution du sudoku

3.2.1 AC3

Nous avons mis en place quelques algorithmes dans la classe *Solver*. Le premier est l'algorithme *AC3* pour vérifier la consistence des arcs. Ici, on considère chaque *Cell* comme une variable ayant pour domaine $1, 2, 3, ..., N$ ou N est la taille du sudoku. Les contraintes sont ici propagées dans la méthode :

```
void Solver::updateSudokuConstraints();
```

La contrainte propagée est qu'il ne doit pas y avoir 2 cases dans la même ligne, la même colonne, la même case avec la même valeur.

3.2.2 Backtracking search + MRV

Pour la résolution en elle-même on a choisit l'algorithme *Backtracking search* amélioré avec *MRV*. On peut le voir dans la méthode :

```
bool Solver::solve();
```

La première étape est de trouver la variable ayant le plus grand nombre de contraintes (et donc le moins de valeurs possibles). Une fois que cette valeur est choisie, on remplace sa valeur dans le sudoku et on commence la construction de l'arbre de résolution. On a donc l'état actuel comme racine d'un noeud, et les possibilités de la MRV en branche. On choisit alors une branche qu'on essaye de résoudre. Si la résolution n'est pas possible, on retourne à l'état actuel et on tente de résoudre une autre branche. Si aucune branche ne peut-être résolue, alors le sudoku n'a pas de solutions. cf Fig 1

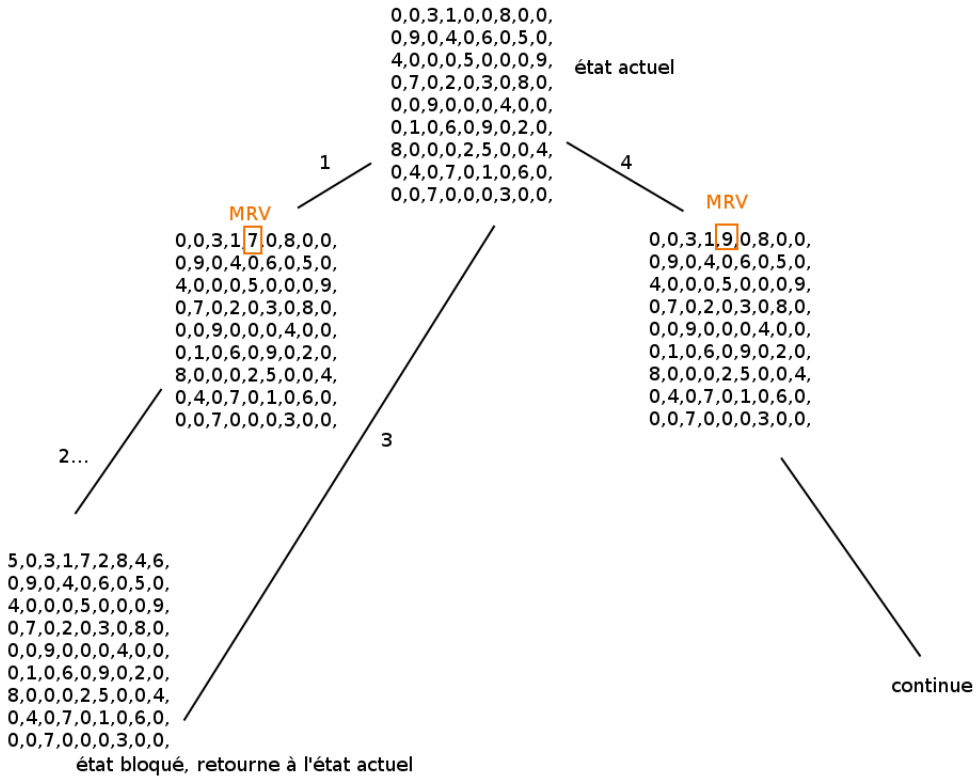

Figure 1: Backtracking + MRV

3.2.3 Forward checking

Enfin, un dernier point a été mis en place. L'exploration s'arrête lorsqu'une valeur n'a plus de valeurs possibles.

4 Conclusion

Ce TP a été l'occasion d'implémenter les différents algorithmes vus en cours comme le *Backtracking + MRV* ou encore *AC3* et donne un exemple de problèmes à satisfaction de contraintes.