

Etape 3 : Aide à la décision

Rapport Projet

CHEUCLE Pierre – HUYNH Pascal– ODIER Valentin – WAKIM Marie

24/05/2012

Contenu

Introduction	2
1ère Version : Vérification des possibilités	Error! Bookmark not defined.
2ème Version : Choix de la case avec le moins de possibilités	Error! Bookmark not defined.
3ème Version : Choix de la case avec le plus de possibilités	Error! Bookmark not defined.

Introduction

Ce projet nous a amenés à réaliser un algorithme permettant de résoudre un sudoku. Pour parvenir à cela nous avons dû imaginer différentes stratégies de recherche telles que :

- *Vérification des possibilités case par case*
- *Affectée la case avec le moins de possibilités*
- *Affectée la case avec le plus de possibilités*
- *Affectée la case avec le moins d'influence sur la suite*
- *...*

Dans une première partie nous rappellerons la structure de données que nous avons choisie, puis nous expliquerons les stratégies développées.

Rappel de notre structure

Nous avons gardé la même structure de données que dans le TP1 :
Par exemple, pour cette grille de sudoku :

4	2		
			1
2			

En utilisant un vector de vector d'entiers. (`Vector<Vector<int>>`), nous avons :

↓	↓	↓	↓
0	4	0	2
0	2	0	0
0	0	0	0
0	0	1	0

Les stratégies

Dans cette partie nous allons développer les stratégies mises en place. Les noms des fonctions sont ceux du programme que nous simplifions par « fonctionX » (avec X le numéro correspondant).

Ci-dessous nous avons simplement retenu les fonctions dont la stratégie est la plus intéressante :

Super_resolution_propagation_contraintes2

La fonction2 a été expliquée dans le tp2 : pour chaque cases vides de la grille de Sudoku, nous calculons le nombre de valeurs possibles. Puis nous choisissons d'affecter la case avec le moins de possibilités.

Cette fonction n'est pas très optimisée, mais nous a été utile pour réaliser les stratégies suivantes.

Super_resolution_propagation_contraintes5

Dans la fonction5, nous affectons une case, puis nous analysons l'impact de ce changement sur les autres cases du sudoku.

Nous conservons alors la case qui aura le moins d'impact sur le reste du sudoku.

Super_resolution_propagation_contraintes8

Dans la fonction 8, nous utilisons le principe utilisé pour la fonction5, nous choisissons d'affecter la case qui :

- A le moins de possibilités
- Aura le plus d'impact sur le reste du sudoku.

Super_resolution_propagation_contraintes4

La fonction4 est semblable à la fonction8, nous affectons la case qui :

- a le moins de possibilités
- aura le moins d'impact sur le reste du sudoku.

Etude comparative des résultats

Notre analyse

Nous avons effectué des tests pour chaque fonction avec différentes grilles de sudoku 3x3. Ces tests nous ont permis de trouver les fonctions les plus et moins performantes.

Pour chaque fonction, nous relevons :

- L'état de changement : le nombre de nœuds parcourus dans l'arbre
- Le nombre de retour : le nombre de remontée dans l'arbre
- Le temps d'exécution : le temps d'exécution de l'algorithme

Nous avons testé chaque fonction sur 14 Sudokus différents :

- Sur 9 sudokus pré-remplis
- Les suivants sont des sudoku vides : 2x2, 3x3, 4x4, 5x5 et 6x6.

Nous obtenons les résultats suivants :

(tableau excel)

Analyse des tests :

Notre première observation permet de voir que les fonctions 6 et 10 sont inefficaces au vue des résultats aberrants obtenus. Nous ne les utiliserons donc pas pour comparer les résultats : les stratégies utilisées ne sont pas efficaces.

Donc analyse entre les fonctions 2, 4, 5, 7 et 8 :

Nous constatons que la fonction 2 est la fonction qui s'exécute la plus rapidement pour la plupart des sudokus (sudokus 1, 2, 3, 4, 8 et 11).

A l'inverse, la fonction 5 est la fonction qui s'exécute la plus lentement pour la plupart des sudokus (sudokus 1, 2, 3, 4, 7, 9, 11) .

Les fonctions 7 et 4 apparaissent aussi comme une bonne fonction.

Finalement, il est difficile de conclure sur quelle est la fonction la plus performante. Cependant des fonctions apparaissent plus performantes que d'autres : les 2, 4, 7 et 8.

Par ailleurs, nous observons qu'une fonction peut parcourir un nombre d'états important et s'exécuter rapidement : ceci étant lié à l'algorithme.