Kevin QUACH
Dylan DE JESUS

Optimisation - TD6 - Init G1 (Mr Fusy)
Université Gustave Eiffel M1 Informatique 2022-2023

Compte Rendu TD6 Optimisation

Sommaire:

Nous avons réalisé le début de ce TP à deux sur les postes de l'université. Certaines tâches ont ensuite été divisées à la maison entre les deux membres du groupe avant de revoir l'intégralité du projet ensemble.

Guide d'utilisateur:

Pour lancer le programme tapez dans le terminal :

Pour la question 11:
python3 sudoku.py [Nom du fichier cnf généré]

Pour la question 12:
python3 sudoku.py [Chemin d'un fichier décrivant une configuration de grille initiale]

Le fichier décrivant la grille doit être au format:
[Taille de la grille]
[Valeurs contenues dans la grille]

La taille de la grille indiquée doit être correcte!

Exemple:

```
4
0 0 0 4
2 0 0 0
0 1 0 0
0 0 1 0
```

Interprétation de la sortie

Sur le terminal:

- Lorsque la grille passée en paramètre de la ligne de commande n’est pas solvable, le programme l’indique à l’utilisateur et se termine.

- Si la grille est solvable, le programme affiche une grille remplie avec une solution gagnante pour le jeu de Sudoku.

Dans un fichier sudoku.out:

- Si la grille n’est pas solvable, le fichier indique “UNSAT”
- Si la grille est solvable, le fichier indique SAT sur sa première ligne et une affectation des variables pour une solution satisfaisante sur la deuxième ligne.

Discussion du problème et résultats

Avec notre programme sudoku.py de la question 12, toutes les grilles proposées sur e-learning admettent au moins une solution.
Pour vérifier que notre programme ne puisse pas résoudre un jeu avec une configuration unsolvable, nous l’avons testé sur des configurations fausses (les fichiers unsolved dans le dossier Grilles).

On constate que le nombre de variables dans la formule cnf est égal à N^3 avec N la taille de la grille de Sudoku. Ce nombre s’explique par le fait qu’il existe une variable pour chaque nombre possible (1 à N) sur chaque case de la grille (N * N).

Pour chaque case, AtLeast renvoie 1 clause tandis que AtMost renvoie (N * (N-1)) / 2 clauses (cf. le cours). Sachant qu’une grille contient N^2 cases, on doit ajouter 1 + (N * (N-1)) / 2 clauses.
De plus, il faut ajouter les clauses pour les règles d’affectation, de lignes, de colonnes et de sous-grilles. Le nombre de clauses est multiplié par 4.
Enfin, il y a les clauses indiquant les cases déjà remplies dans la configuration initiale.

Nombre total de clauses: ((1 + (N * (N-1)) / 2) * 4 + p)
Avec p le nombre de cases pré-remplies.

Pour calculer les temps de génération de clauses et de résolution, on a ajouté des instructions à cet effet dans sudoku.py de la question 12:

- Sur une grille 4 x 4:
  - Génération de clauses: 0.000289 secondes
  - Résolution du problème : 0.002476 secondes
- Sur une grille 8 x 8:
  - Génération de clauses: 0.004710 secondes
  - Résolution du problème: 0.004532 secondes
- Sur une grille 16 x 16:
  - Génération de clauses: 0.065140 secondes
  - Résolution du problème: 0.021408 secondes
- Sur une grille 25 x 25:
  - Génération de clauses: 0.488777 secondes
  - Résolution du problème: 0.100026 secondes

On constate que le temps de génération croît plus vite que le temps de résolution. Cela pourrait s'expliquer par le fait que la fonction de génération de clause a une complexité de plus grande ordre de grandeur que celle qui résout le problème.

Ce que nous avons appris:

- Utiliser **minisat**
- Lire et comprendre un fichier cnf
- Interpréter et comprendre la sortie de **minisat**
- Formaliser un problème écrit en français en format cnf
- Écrire un programme qui automatise la résolution d'un problème
- Impressionner nos amis en résolvant un Sudoku 25x25 rapidement

Difficultés rencontrés

Utiliser les solutions fournies par **minisat** et les représenter dans une grille.
Quelques problèmes avec l'affichage de la grille, elle était initialement renversée.