Université Gustave Eiffel

Master Informatique

Rapport Technique

Optimisation

Jimmy Teillard

Groupe 2

Chargé de TD: Anthony Labarre

1ère Année

Décembre 2021

Contents

1 Introduction

Ce rapport technique à pour but de compiler mes résultats quant aux exercices du TP minisat.

Chacune des parties du TP sera traitée dans une section dédiée, avec les résultats et réponses aux questions posées, en plus d'un mode d'emploi du script créé.

2 Mode d'emploi

Utilisation : `python3 sudoku.py <input.txt> [-c]`

Pour lancer le script, il est nécessaire de fournir un fichier en entrée (ex: `input.txt`) formaté correctement, représentant une grille de Sudoku à résoudre.

Il est possible d'ajouter une option `-c` afin de supprimer les fichiers intermédiaires générés pour calculer la solution.

Voici un exemple de fichier d'entrée valide :

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

La première ligne contient une valeur qui correspond à la dimension de la grille à résoudre (et, en particulier, les valeurs possibles qu'une case peut prendre). Par exemple, si la valeur est 4, alors la grille est de 4 cases par 4 cases, chacune pouvant prendre une valeur de 1 à 4 inclus.

Les lignes suivantes représentent chacune la grille de départ, qui est à résoudre. Chaque case est symbolisée par un nombre (les nombres sont séparées par des espaces, et chaque retour à la ligne représente une nouvelle ligne de la grille). Une case vide est représentée par un 0.

Lancer le script permet de lancer minisat sur ces données après qu'elles aient été traités de sorte à les formuler comme un problème SAT ; le résultat est formaté de la même façon que le fichier d'entré : c'est la grille résolue.

Dans le cas où la grille de départ n'est pas solvable, le message suivant apparaît :

"This grid is unsolvable"

3 Explications

Je préface mon analyse par le fait que mon programme fonctionne correctement (c'est à dire qu'il **semble** pouvoir résoudre n'importe quelle grille, soumettant un temps potentiellement tendant vers l'infini, du moment qu'elle soit solvable).

Bien qu'au début, je me contentait principalement de suivre les consignes, au fur et à mesure du développement du programme, j'ai appris à comprendre comment former une entrée valable au problème SAT à partir des informations reçues, notamment comment créer des équivalences à partir d'idées telles que "au moins un" et "au plus un".

Il est alors à noter que dans le cas d'une grille de Sudoku, la ramener à un problème SAT revient à d'abord créer exactement N^3 variables, où N est la dimension de la grille / le nombre de valeurs possibles. En effet, cette idée correspond au fait qu'il y ait N^2 cases qui peuvent chacune prendre N valeurs.

Un problème qui est survenu lors du développement est l'idée de retourner à une expression littérale (un tuple contenant le signe de la variable, sa position sur la grille, et sa valeur) à partir de l'entier qui représente ladite variable. Ainsi, j'ai d'abord tenté de retirer chaque membre du tuple (excepté le premier, le signe) comme étant des coordonnées en 3 dimensions, en vain. Je me suis donc rabattu sur le principe plus simple qui est de garder un dictionnaire liant une variable SAT à son littéral. J'ai pu donc me servir dudit dictionnaire pour reconstituer la grille solution à partir de la sortie de minisat.

Mon programme fonctionne effectivement sur toutes les grilles fournies sur elearning :

- Les grilles 4x4 ont 64 variables, et 448 + x clauses (où x est le nombre de valeurs déjà entrées). La résolution est instantanée.
- Les grilles 9x9 ont 729 variables, et 11988 + x clauses (où x est le nombre de valeurs déjà entrées). La résolution est instantanée.
- Les grilles 16x16 ont 4096 variables, et 123904 + x clauses (où x est le nombre de valeurs déjà entrées). La résolution prend environ 0.35s.
- Les grilles 25x25 ont 15625 variables, et 752500 + x clauses (où x est le nombre de valeurs déjà entrées). La résolution prend environ 2s.

On peut facilement conjecturer, au vu du temps et du nombres de clauses croissants, que le problème est très difficile, NP-difficile. Vérifier qu'une solution est valable est en revanche très simple en temps polynomial. Le problème est NP-complet.

Finalement, pour m'amuser, j'ai tenté de résoudre des grilles fournies par une application de Sudoku sur mon smartphone, avec succès.