TP5 – Préparation au projet

Projet de programmation M1

4 et 16 Novembre 2014

1 Le problème SAT

1.1 Un problème difficile mais utile

Le problème SAT est un problème qui consiste à trouver une solution à un système de contraintes. Plus précisément, on définit une formule F sur un ensemble de variables booléennes (*i.e* de domaine $\{0,1\}$) $\{x_1,\ldots,x_n\}$ en donnant une suite de contraintes (autrement appelés *clauses*) $\{C_1,\ldots,C_m\}$. Un littéral est soit une variable, soit sa négation (notée $\neg x$). Une clause est nécessairement une disjonction de littéraux. Par exemple, en notant $\vee$ le "ou" booléen :

$$x_1 \vee \neg x_2 \vee x_3 \vee x_4$$

Un exemple de formule ainsi construite (en notant $\wedge$ le "et" booléen) :

$$F = (x_1 \vee \neg x_2 \vee x_3 \vee x_4) \wedge (\neg x_1 \vee x_2) \wedge (x_3 \vee \neg x_4)$$

Le problème SAT est le problème de trouver, étant donné une formule F ainsi construite, une solution de F. Pour l'exemple ci-dessus, on a entre autre $x_1 = 1, x_2 = 1, x_3 = 0, x_4 = 0$ qui est une solution. Mais aussi $x_1 = 0, x_2 = 1, x_3 = 1, x_4 = 0$. En revanche, $x_1 = 0, x_2 = 1, x_3 = 0, x_4 = 0$ n'est pas solution puisque cela ne satisfait pas la première clause.

On ne connaît pas à ce jour d'algorithme théoriquement efficace pour résoudre SAT. Par efficace, on entend dont le nombre d'opérations effectuées pour résoudre le problème est polynomial en la taille de l'entrée. En fait, on soupçonne fortement que de tels algorithmes n'existent pas mais on ne sait toujours pas le démontrer (c'est la fameuse question à 10^6 dollars "$\mathsf{P} = \mathsf{NP}$?"). Ces soupçons sont renforcés par le fait que le problème SAT est équivalent à de nombreux autres problèmes, provenant de domaines très variés, qui semblent tous difficiles à résoudre. L'expressivité du problème SAT permet beaucoup d'applications différentes. Dans ce TP, on va voir comment on peut par exemple exprimer le problème du Sudoku avec SAT. Pour le projet, vous aurez à traduire des contraintes en un problème SAT pour créer des emplois du temps.

1.2 Le SAT solver Minisat

La difficulté théorique apparente du problème SAT n'a pas empêché le développement de logiciels efficaces en pratique. En effet, la plupart des instances pratiques de ce problème proviennent de contraintes liées à des systèmes physiques réels et contiennent donc certaines structures implicites qui peuvent être exploitées pour résoudre SAT efficacement. On appelle de tels logiciels des SAT solvers. Les plus performants aujourd'hui sont capables de résoudre des problèmes contenant quelques millions de variables et quelques centaines de milliers de clauses. Dans ce TP et pour votre projet, nous allons regarder un SAT solver en particulier : Minisat. C'est un solver qui se veut simple (code assez réduit) mais efficace. Il est gratuit, open source et téléchargeable à l'adresse `http://www.minisat.se/`. Son utilisation est la suivante : on lui donne en entrée un fichier décrivant la formule F et un fichier de sortie. Minisat tente de résoudre le problème. Si il y parvient, il écrira dans le fichier de sortie "SAT" et une solution. Par exemple, pour la solution $x_1 = 1, x_2 = 1, x_3 = 0, x_4 = 0$, il écrira :

```
SAT
1  2  -3  -3
```

Si la formule n'est pas satisfiable, il écrira tout simplement "UNSAT".

Le format d'entrée pour décrire une formule est très élémentaire. C'est le format DIMACS. On commence par écrire toujours la même chose :

```
p cnf n m
```

où n est le nombre de variables et m le nombres de clauses de la formule. Ensuite, on décrit chaque clause sur une ligne du fichier, en indiquant la liste de ses variables, avec un $-$ devant si la variable est niée. On termine la ligne par un 0 et un saut de ligne. Par exemple, la formule F donnée dans la section 1.1 sera décrite par le fichier :

```
p cnf 4 3
1 -2 3 4 0
-1 2 0
3 -4 0
```

Exercice 1. Familiarisez-vous avec Minisat en essayant l'exemple ci-dessus. Vous pouvez en essayer d'autres. Il faut entrer dans le terminal la commande `minisat entree sortie`.

2 Résoudre les Sudokus

On se propose d'utiliser Minisat pour résoudre les Sudokus. Pour cela, nous allons traduire une grille de Sudoku en une formule CNF qui sera satisfiable si et seulement si la grille de départ était correcte. De plus, on pourra, à partir d'une solution de cette formule, reconstruire la grille complétée. On n'aura donc qu'à créer un programme C qui étant donné une grille de Sudoku partiellement remplie traduit cela en un fichier au format DIMACS, puis on exécutera Minisat sur ce fichier et on extraira une solution à la grille de la solution ainsi calculée.

2.1 Représentation d'un Sudoku

On représentera une grille partiellement remplie de Sudoku par un fichier texte ayant 9 lignes de 9 chiffres. Chaque ligne du fichier représente une ligne de la grille, chaque caractère de cette ligne est un élément de cette ligne. On indique qu'une case est vide en lui donnant la valeur 0. Par exemple, la grille de la figure 1 sera représentée par le fichier :

```
853604002
904030600
060000034
000000490
500000007
079000000
310000060
005070301
700103549
```

Exercice 2. Écrire une fonction `filetogrid(char *filename, int grid[9][9])` qui transforme un fichier en une grille 9×9. On adopte la même convention que pour le fichier : une case vide est représentée par un 0.

2.2 Des grilles aux CNFs

Pour représenter une grille par une CNF, nous allons introduire pour chaque case (i, j) de la grille et pour chaque valeur $k \in \{1, \ldots, 9\}$ une variable $v_{i,j}^k$. On veut intuitivement que dans la formule qu'on va construire, $v_{i,j}^k = 1$ signifie que la case (i, j) contient la valeur k. Plus formellement, étant donné une grille G de Sudoku partiellement remplie, on veut construire une formule F_G sur les variables $v_{i,j}^k$ telle que toute assignation des variables satisfaisant F_G vérifie :

– si dans G la case (i, j) contient la valeur k, alors $v_{i,j}^k = 1$.

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

FIGURE 1 – une grille de Sudoku

- pour chaque case (i, j), il existe exactement une valeur k telle que $v_{i,j}^k = 1$. Autrement dit, chaque case contient exactement une seule valeur.
- la grille telle que toute case (i, j) contient la seule valeur k telle que $v_{i,j}^k = 1$ est une grille de Sudoku valide.

Exercice 3. Dans cet exercice, nous détaillons les différents constituants de la formule F_G :

1. Écrire une formule CNF A sur les variables $v_{i,j}^k$ telle que A est vraie si et seulement si pour chaque (i, j), il existe k tel que $v_{i,j}^k = 1$.
2. Écrire une formule CNF $U_{i,j}$ telle que $U_{i,j}$ est vraie si et seulement si il y a au plus un k tel que $v_{i,j}^k = 1$.
3. Écrire une formule CNF $L_{i,k}$ telle que L_i est vraie si et seulement si il existe j tel que $v_{i,j}^k = 1$.
4. En déduire une formule CNF L_i qui est vraie si et seulement si pour chaque $k \in \{1, \ldots, 9\}$, il y a j tel que $v_{i,j}^k = 1$.
5. Même question pour C_j qui expriment que pour tout k, il existe i tel que $v_{i,j}^k = 1$.
6. Même question pour Z_r qui exprimer que dans une des zones 3×3 du Sudoku on trouve toutes les valeurs de 1 à 9.
7. Assembler tout ça pour transformer une grille G en une CNF F_G dont les solutions correspondent à des remplissages valides de la grille G.

Exercice 4. Écrire un programme qui étant donné en entrée un fichier contenant une grille de Sudoku G, produit un fichier au format DIMACS décrivant F_G. On pourra représenter la variable $v_{i,j}^k$ par $i \times 100 + j \times 10 + k$ par exemple.

Écrivez un programme qui prend en entrée le fichier réponse de Minisat sur votre représentation de F_G et affiche la grille complétée correspondante.

Utilisez votre programme pour résoudre le Sudoku de la figure 1.