Complexité TP2 : Réductions

Université d'Aix-Marseille, M1 Informatique

Année universitaire 2020–2021

Préambule : Dans ce deuxième travail à réaliser, 3 exercices (des "mini-projets") assez simples sont à traiter, de préférence dans l'ordre (sauf indication contraire de vos enseignants). Le langage de programmation recommandé est le langage C mais vous n'y êtes pas astreints. Pour chaque mini-projet, vous devrez bien sûr rédiger un petit compte rendu qui décrira notamment les algorithmes implémentés et leur complexité et fournira tout élément nécessaire à la prise en main des programmes correspondants, ainsi bien évidemment que le code (les détails seront précisés en TP).

1 Mini-projet 1. Vérificateur déterministe pour SAT

Le problème SAT étant dans la classe **NP**, il existe un algorithme polynomial non déterministe pour établir si une formule en forme normale conjonctive (FNC) possède une affectation qui la satisfait et, de façon équivalente, il existe aussi un vérificateur déterministe polynomial pour SAT, qui prend en entrée une formule en FNC et une affectation pour ses variables et vérifie si l'affectation satisfait la formule.

Implémentez un vérificateur déterministe pour SAT qui travaille sur des fichiers en format *DIMACS CNF* comme le suivant :

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

La première ligne du fichier commence par `p cnf` suivi par le *nombre de variables* de la formule (ici 3), suivi par le *nombre de clauses* (ici 2). Chacune des lignes suivantes représente une clause de la formule ; les nombres positives et négatives représentent les variables (numérotées à partir de 1) en forme positive ou négative, et le `0` termine la clause (il n'y a pas de variable numéro 0). Donc ce fichier d'exemple représente la formule $(x_1 \vee \neg x_3) \wedge (x_2 \vee x_3 \vee \neg x_1)$.

Au delà du fichier contenant la formule, le programme reçoit aussi en entrée un deuxième fichier qui contient l'affectation aux variables, dans le format suivant :

```
1 -2 -3
```

Ce fichier ne contient qu'une ligne où toute les variables apparaissent exactement une fois soit en forme positive, soit en forme négative, et représente donc l'ensemble de vrais littéraux. Ce fichier d'exemple représente donc l'affectation $x_1 = 1, x_2 = 0, x_3 = 0$.

2 Mini-projet 2. Réduction de Zone Vide à SAT

Le problème SAT étant **NP**-complet, pour chaque problème dans **NP** il existe une réduction polynomial vers SAT. Une solution efficace pour SAT permet donc de résoudre de forme potentiellement efficace n'importe quel autre problème dans **NP**. Par conséquent, beaucoup de travail

a été effectué pour réaliser des logiciels qui permettent de résoudre de la façon la plus efficace possible le problème SAT, des *SAT solvers*.

Pour ce projet on va utiliser le solver MiniSat, qui est disponible sur la page web `http://minisat.se/MiniSat.html`. MiniSat utilise en entrée le format DIMACS CNF décrit dans le mini-projet 1. En exécutant MiniSat sur un fichier en format DIMACS CNF (par exemple, `minisat formule.txt`) on obtient soit une affectation qui satisfait la formule, soit une indication du fait que la formule n'est pas satisfaisable.

Implémentez une réduction à partir de cette variante du problème de la zone vide vers SAT :

> **Données :** Un graphe non-orienté sans boucle $G = (S, A)$, et un entier positif $k \leq |S|$.
> **Question :** Existe-t-il dans G une zone vide de taille k ou plus ?

Le programme devra donner en sortie une formule en format DIMACS CNF, qui verra ensuite donnée en entrée à MiniSat.

Implémentez également un algorithme « force brute » pour résoudre ce même problème, qui après avoir effectuée la réduction essaye toutes les affectations possibles en vérifiant s'il y en a une qui satisfait la formule avec la procédure implémentée dans le mini-projet 1. Enfin, comparez le temps d'exécution des deux solution (réduction + SAT solver vs réduction + algorithme force brute) pour des entrées de taille suffisamment grande.

3 Mini-projet 3. Réduction de Sudoku à SAT

Le jeu Sudoku consiste en une grille de jeu carrée de neuf cases de côté, subdivisé en autant de sous-grilles carrées identiques, appelées « régions ». Le but du jeu est de remplir cette grille avec des chiffres allant de 1 à 9 en veillant toujours à ce que chaque ligne, chaque colonne et chaque région ne contiennent qu'une seule fois tous les chiffres allant de 1 à 9. Au début du jeu, un certain nombre de chiffres sont déjà placés.

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

Considérons la version *généralisée* suivante du jeu Sudoku :

> **Données :** Une grille de taille $n^2 \times n^2$ où chaque case peut contenir un entier dans l'intervalle $[1, n^2]$, ou bien elle peut être vide.
> **Question :** Existe-t-il une façon de remplir les cases vides de la grille avec des entiers dans l'intervalle $[1, n^2]$ telle que chacune des n^2 lignes, colonnes et régions de la grille ne contient qu'une fois tout les entiers de l'intervalle ?

Implémentez une réduction de Sudoku vers SAT, qui donne en sortie une formule en format DIMACS CNF pour MiniSat, et vérifiez expérimentalement le temps d'exécution de la réduction plus le temps d'exécution du SAT solver sur des instances de taille croissante.