TP7 – Un petit SAT Solver

Projet de programmation M1

02 Décembre 2014

Dans ce TP, on se propose de programmer un modeste SAT Solver. Le but est d'essayer d'implémenter une version basique de l'algorithme DPLL et d'essayer des méthodes probabilistes.

Les variables de la formule seront représentées par des entiers strictement positifs et les littéraux (une variable ou sa négation) par un entier positif si la variable apparaît positivement et par un entier négatif sinon. Par exemple, le littéral $\neg x_1$ sera représenté par -1 alors que x_1 sera représenté par 1. On représente une clause par la structure suivante :

```
struct clause {
  int size;
  int *lit;
};
```

Par exemple, la clause $x_1 \vee \neg x_2 \vee x_3$ sera représentée par la structure :

```
{
  .size = 3,
  .lit = [1,-2,3]
}
```

Une formule sera représentée par un tableau de clauses. Une assignation σ des variables $x_1, \ldots, x_n$ dans $\{0, 1\}$ sera représentée par un tableau t de taille n tel que pour $i \in \{0, \ldots, n-1\}$, $t[i] = \sigma(x_{i+1})$.

Exercice 1. Écrire une fonction `sat_clause` qui étant donné une clause et une assignation, renvoie 1 si la clause est vraie sur cette assignation et 0 sinon. En déduire une fonction `sat_formula` qui étant donné une formule et une assignation, renvoie 1 si la formule est satisfaite par cette assignation (c'est-à-dire que toutes ses clauses le sont) et 0 sinon.

Exercice 2. On autorise maintenant les assignations à être partielles (toutes les variables n'ont pas forcément de valeur). On représente cela par la valeur -1. Par exemple, si x_1 n'est pas assigné, on a $t[0] = -1$. Modifier la fonction `sat_clause` pour qu'elle renvoie -1 si la clause n'est pas satisfaite par une assignation partielle mais que toutes ses variables ne sont pas encore assignées.

On rappelle l'algorithme DPLL. Étant donné une formule F :
- Choisir x parmi les variables de F.
- Exécuter $DPLL(F[x \leftarrow 1])$. Si $F[x \leftarrow 1]$ est satisfaite par σ, renvoyer la solution $\sigma \cup \{x \mapsto 1\}$.
- Sinon exécuter $DPLL(F[x \leftarrow 0])$. Si $F[x \leftarrow 0]$ est satisfaite par σ, renvoyer la solution $\sigma \cup \{x \mapsto 0\}$. Sinon, renvoyer que F n'est pas satisfiable.

Par exemple, prenons la formule $F = (x \vee y \vee \neg z) \wedge (\neg x \vee \neg z) \wedge (\neg z \vee \neg y) \wedge (z \wedge \neg x)$. On choisit d'abord $z = 1$. On a alors $F[z \leftarrow 1] = (x \vee y) \wedge \neg x \wedge \neg y$. On choisit $x = 1$: $F[z \leftarrow 1][x \leftarrow 1]$ n'est pas satisfiable. Donc on choisit $x = 0$. $F[z \leftarrow 1][x \leftarrow 0] = y \wedge \neg y$. Qu'on choisisse $y = 1$ ou $y = 0$, la formule n'est pas satisfiable. Donc on choisit $z = 0$. On a alors $F[z \leftarrow 0] = \neg x$. On choisit $x = 1$ et on a une solution de F.

En fait, dans DPLL, on construit petit à petit une solution potentielle. Dès qu'on est confronté à une absurdité, on revient à la dernière décision prise et on l'annule.

Exercice 3. Implémentez cette version de l'algorithme DPLL (en récursif). On choisira les variables dans l'ordre de leur numérotation et on utilisera bien sûr les fonctions de l'exercice précédent !

On peut facilement améliorer cet algorithme en ajoutant deux ingrédients :

– *unit clause propagation* : si une clause n'a plus qu'une seule variable non-assignée, alors on sait quelle valeur il faut choisir pour cette variable. Par exemple $C = \neg x \vee y$ et y est déjà assigné à 0, alors on sait qu'il faut avoir $x = 0$ pour satisfaire C.
– *pure literal elimination* : si une variable x apparaît dans toutes les clauses avec le même signe, alors on sait quelle valeur lui assigner. Par exemple, si toutes les clauses qui contiennent la variable x contiennent le litéral $\neg x$, alors on peut choisir $x = 0$ sans changer la satisfiabilité de la formule.

Exercice 4. Modifier votre fonction DPLL pour qu'avant de choisir la variable suivante, elle effectue les deux simplifications précédentes pour choisir la variable.

Exercice 5. Écrire un programme qui prend en entrée un fichier au format DIMACS (voir TP 5) et en cherche une solution avec l'algorithme DPLL que vous venez de programmer. Essayez-le sur votre Sudoku !

S'il vous reste du temps, vous pouvez essayer de résoudre SAT par des méthodes probabilistes :
– Choisir une assignation aléatoire σ des variables uniformément.
– Si toutes les clauses sont satisfaites, alors renvoyer la solution.
– Sinon, choisir une variable x et essayer la solution σ'_x où $\sigma'_x(x) = 1 - \sigma(x)$ et $\sigma'_x(y) = \sigma(y)$ pour $x \neq y$.

Deux méthodes existent pour choisir la variable x à changer :
– GSAT : choisir la variable x qui minimise le nombre de clauses non satisfaite par σ'_x.
– WalkSAT : choisir aléatoirement une clause C non satisfaite par σ. Choisir x la variable de C qui minimise le nombre de clauses satisfaites par σ mais pas par σ'_x.

Exercice 6. Implémentez GSAT et WalkSAT. Essayez-les avec le Sudoku. Quel est le problème ici ?