TP1 : SAT et l'algorithme DPLL

Optimisation Combinatoire Avancée

Février-Mars 2020

L'objectif des 2,5 premières séances de TP est d'implémenter un solveur SAT et de l'évaluer sur différentes instances du problème. Vous devrez déposer votre programme sur Moodle au plus tard la veille de l'évaluation. Vous pouvez écrire votre programme dans l'un des langages suivants : java, python3. (Si vous voulez utiliser un autre langage, parlez-en à votre enseignant.)

1 Rappel de l'algorithme DPLL

C'est une approche de type *branch and bound* pour décider si un ensemble de clauses $\mathcal{C}$ est satisfiable ou non. On explore un arbre binaire énumérant des interprétations possibles :

1. on initialise la recherche à la racine avec $\mathcal{C}$;
2. à chaque nœud :
   (a) on simplifie si possible l'ensemble de clauses
   cf clauses unitaires / littéraux purs ci-dessous
   (b) on choisit une variable restante
   $\Rightarrow$ 2 valeurs possible vrai / faux $\Rightarrow$ 2 sous-arbres

propagation des clauses unitaires chaque clause à un seul littéral l est supprimée, et toutes les clauses contenant ce littéral sont supprimées, et le littéral opposé est enlevé de toutes les autres clauses (implicitement, ce littéral est rendu vrai) ;

élimination des littéraux purs si une variable n'a que des occurrences positives ou que des occurrences négatives, toutes les clauses la contenant sont supprimées (implicitement, on choisi la valeur de cette variable pour rendre vraies toutes ses occurrences).

Finalement :

- si une clause vide apparaît à un nœud alors échec dans cette branche ;
- si au contraire il n'y a plus de clause, on a une branche de succès.

2 Implémentation itérative

Dans les applications pratiques de SAT, les ensembles de clauses peuvent être très grands, ainsi que les nombres de variables. On implémente donc la recherche sous forme itérative, avec une pile contenant les affectations partielles courantes. Plus précisément, on peut empiler des triplets (X, v, v'), où X est une variable, v est la valeur (vrai ou faux) affectée à X, et v' est éventuellement une autre valeur possible pour X en cas de retour arrière – en choisissant $v' = \text{None}$ s'il n'y a pas d'autre valeur à essayer pour X.

L'ensemble de clauses $\mathcal{C}$ n'est pas physiquement modifié durant la recherche.

Algorithme backtrack – avec pile

1. $n \leftarrow$ nb vars ; fini $\leftarrow$ faux ; $S \leftarrow \emptyset$; . . . . . // S = pile des affectations courantes
2. Tant que pas fini :
   (a) si consistant S :
      i. si $|S| = n$: fini $\leftarrow$ vrai ; . . . . . // *on a trouvé un modèle de* $\mathcal{C}$
      ii. sinon : choisir (X, v, v') ; empiler (X, v, v') sur S
   (b) sinon : . . . . . // S *inconsistant* $\Rightarrow$ *backtrack*
      i. $(X, v, v') \leftarrow$ dépiler S ; . . . . . // S *inconsistant* $\Rightarrow S \neq \emptyset$
      ii. tant que $|S| > 0$ et $v' =$ None : $(X, v, v') \leftarrow$ dépiler S ;
      iii. si $v' \neq$ None : empiler $(X, v',$ None$)$ sur S ;
      iv. sinon fini $\leftarrow$ vrai ; . . . . . // *cas où* $S = \emptyset$, *plus d'autre choix possible*
3. retourner S. // *Si fini en 2(a)i : S contient un modèle ; si fini en 2(b)iv : $S = \emptyset \Rightarrow$ inconsistant*

Explications

Les tests de consistance s'entendent par rapport à l'ensemble de clauses $\mathcal{C}$:

- S est consistant par rapport à $\mathcal{C}$ si S ne rend aucune clause de C fausse ;
- une clause c est fausse pour S si pour chaque littéral l de c : soit $l = X$ et S contient une affection $(X, -1, v')$; soit $l = \neg X$ et S contient une affectation $(X, +1, v')$.

En 2(a)ii, on choisit une nouvelle affectation de valeur v pour une variable X :

- soit par application de la règle de la clause unitaire (alors $v' =$ None) ;
- soit par application de la règle du littéral pur (là encore $v' =$ None) ;
- soit en choisissant une variable non encore affectée et en lui choisissant une valeur v, et en mémorisant l'autre valeur possible v' en cas de retour arrière.

En 2(b)i et 2(b)ii, on «backtrack», c'est-à-dire qu'on défait / dépile les affectations courantes, tant qu'il n'y a pas d'autre choix possible ($v' =$ None).

3 Travail à faire

Préparation des données : il vous faut commencer par écrire une fonction qui calcule la liste des variables qui apparaissent dans $\mathcal{C}$.

Tests de consistance : comme on l'a dit plus haut, $\mathcal{C}$ n'est pas modifié durant l'exécution de l'algorithme. À chaque itération, en 2a, on test si le modèle partiel courant, décrit par S, est consistant avec $\mathcal{C}$. Essentiellement, ça se fait en parcourant une fois $\mathcal{C}$. Vous devez implémenter une fonction test_consistance($\mathcal{C}$, S) qui réalise ce test.

Choix de variables / valeurs : en 2(a)ii, on étend le modèle partiel courant avec une nouvelle affectation (X, v) (v' mémorise s'il reste une autre possibliité à explorer plus tard pour X, en cas de backtrack).

Dans un premier temps, vous pouvez implémeter une fonction de choix simple, qui retourne la première variable non encore affectée dans S.

Dans un second temps, vous devez implémenter une fonction de choix qui sélectionne d'abord les couples (X, v) correspondant à des clauses unitaires de $\mathcal{C}$ – selon le modèle partiel courant. Ces clauses unitaires peuvent être trouvées par la fonction qui réalise le test de consistance en parcourant $\mathcal{C}$. On pourra ajouter la sélection des littéraux purs.

Dans un troisième temps, on s'intéressera à l'*heuristique de choix des variables de décision* – lorsqu'on doit choisir une variable pas encore affectée, et que ce choix n'est pas conséquence d'une clause unitaire

ni d'un littéral pur. Il faut alors choisir une «bonne» variable de décision. L'idée générale des heuristiques de sélection est de privilégier les variables qui permettent de :

- satisfaire un maximum de clauses non encore satisfaites ;
- raccourcir le plus possible de clauses déjà courtes, afin d'essayer d'arriver rapidement à de nouveaux littéraux purs.

Calcul de modèles partiels : dans l'algorithme ci-dessus, on n'a un succès (en 2(a)i) que lorsqu'on a un modèle complet. Mais l'évalutation de $\mathcal{C}$ avec un S partiel peut révéler qu'un ensemble de clauses est déjà satisfait avec un modèle partiel. On peut modifier l'algorithme pour qu'il retourne un tel modèle partiel, en changeant la condition d'arrêt en 2(a)i. L'intérêt est qu'on obtient alors une solution plus générale.

Tests : Pour la première séance, on pourra faire des tests sur les petites instances vues en cours et TDs. Toutefois, pour se conformer au standard DIMACS pour le codage des instances de SAT, on représentera les variables par des nombres entiers > 0, et les littéraux par des entiers signés. Ainsi :

$$[[1, -2, -3, 4], [-1, -3, 4], [3, 4], [-4]$$

pourra représenter le premier ensemble de clauses ci-dessous.

1. $\{p \vee \neg q \vee \neg r \vee s\ ,\ \neg p \vee \neg r \vee s\ ,\ r \vee s\ ,\ \neg s\}$
2. $\{p \vee \neg q \vee \neg r \vee s\ ,\ \neg p \vee \neg r \vee s\ ,\ r \vee s\ ,\ \neg s\ ,\ p \vee q\}$
3. $\left\{ \begin{array}{c} p \vee q \vee r\ ,\ p \vee q \vee \neg r\ ,\ p \vee \neg q \vee r\ ,\ p \vee \neg q \vee \neg r\ , \\ \neg p \vee s\ ,\ p \vee \neg s \vee \neg u\ ,\ \neg p \vee v \end{array} \right\}$
4. $\{p \vee q \vee r\ ,\ \neg p \vee \neg q \vee \neg r\ ,\ \neg p \vee q \vee r\ ,\ \neg q \vee r\ ,\ q \vee \neg r\}$
5. $\{p \vee q \vee r\ ,\ p \vee \neg q\ ,\ q \vee \neg r\ ,\ r \vee \neg p\ ,\ \neg p \vee \neg q \vee \neg r\}$
6. $\{\neg p \vee q \vee r, \neg p \vee \neg q \vee \neg r\ ,\ p \vee \neg q \vee r\ ,\ p \vee \neg q \vee \neg r, \neg p \vee s\ ,\ p \vee \neg s \vee \neg t \vee \neg u\ ,\ \neg p \vee v\}$
7. $\left\{ \begin{array}{c} p \vee \neg r \vee t\ ,\ p \vee s\ ,\ \neg p \vee \neg q \vee \neg r\ ,\ \neg p \vee \neg q \vee s \vee \neg u\ ,\ \neg p \vee \neg s \\ q \vee \neg u\ ,\ \neg q \vee t \vee u\ ,\ r \vee s \vee u\ ,\ \neg r \vee \neg s \vee \neg t \vee \neg u\ ,\ s \vee u \end{array} \right\}$

4 Application à la coloration de graphe

Étant donné un graphe, et un nombre de couleur fixé k, on peut se demander s'il est possible de trouver un coloriage des sommets du graphe avec ces k couleurs sans que 2 sommets soient de la même couleur. On vous demande de tester votre solveur SAT sur des problèmes de coloration de graphe.

Pour représenter un tel problème de coloration par des clauses, on pourra introduire une variable x_{ij} pour chaque sommet i et chaque couleur possible j (donc pour $j \in \{1, \ldots k\}$ si on s'autorise k couleurs). Il faudra introduire des clauses pour :

- représenter le fait que chaque sommet a une couleur, et une seule ;
- représenter la contrainte que deux sommets adjacents n'ont pas la même couleurs.

Des instances de graphes à colorer sont sur Moodle : les sommets sont représentés par des entiers, et un fichier donne une arête par ligne. Pour k, on pourra essayer avec des valeurs assez grandes, en fonction des inidcations données dans les commentaires en début de fichier.

Vous devrez donc écrire un programme qui lit un graphe dans un de ces fichiers, prend en paramètre le nombre de couleurs k qu'on s'autorise, traduit cette instance en une instance de SAT, et lance votre solveur

pour décider si une telle coloration avec k couleurs est possible ; si oui, il faut écrire la coloration dans un fichier : une ligne par sommet, sur chaque ligne le numéro du sommet et sa couleur – où les couleurs seront représentées par des entiers naturels.

5 Évaluation

Vous devez déposer sur Moodle une archive contenant :

- les sources de votre programme ;
- un exécutable pour lancer le TP. Un menu ou des options devront être disponibles pour pouvoir tester les différentes parties du TP ;
- un "readme" expliquant comment on utilise cet exécutable ; une ligne par variable, et deux colonnes : les numéros des variables dans la première colonne, leurs valeurs, -1 ou 1, dans la seconde.

L'évaluation aura lieu en séance, durant 30mn : vous devrez exécuter lancer votre programme sur des instances dévoilées au moment de l'évaluation, et reporter vos résultats sur une fiche distribuée pour cela.