DM : Problème MAX-SAT

8 mars 2025

1 Définitions

Le problème MAX-SAT est le problème d'optimisation défini de la façon suivante :

- **Entrée :** une formule φ sous forme normale conjonctive
- **Sortie :** Une valuation v maximisant le nombre de clauses satisfaites dans la formule φ.

On associe MAX-SAT au problème de décision (problème de seuil) suivant :

- **Entrée :** Une formule φ sous forme normale conjonctive, un entier C
- **Sortie :** Existe-t-il une valuation v satisfaisant au moins C clauses dans φ ?

Pour un entier K donné, il est également possible de définir le problème MAX-K-SAT, qui correspond à la restriction de MAX-SAT aux instances suivantes :

- **Entrée :** une formule φ sous forme normale conjonctive dont les clauses contiennent au plus K littéraux
- **Sortie :** Une valuation v maximisant le nombre de clauses satisfaites dans la formule φ.

De même, au associe ce problème à un problème de seuil (correspondant au problème de seuil précédent restreint aux FNC dont les clauses ont au plus K littéraux).

2 NP-Complétude de MAX-2-SAT

MAX-SAT est un problème NP-Complet. Pour le montrer, on peut montrer une propriété plus forte, qui est que MAX-K-SAT est NP-complet pour tout $K \geq 2$.

Pour montrer que MAX-2-SAT est NP-Difficile, on utilise une réduction polynomiale depuis 3-SAT.

On considère trois littéraux l_1, l_2 et l_3 et une variable x indépendante des trois littéraux : on note $\overline{l_1}$ la négation de l_1 (on a donc, pour une variable x_1, $\overline{x_1} = \neg x_1$ et $\overline{\neg x_1} = x_1$). Pour ces trois littéraux, on considère la conjonction des dix clauses ci-dessous :

$$l_1 \wedge l_2 \wedge l_3 \wedge x \wedge (\overline{l_1} \vee \overline{l_2}) \wedge (\overline{l_2} \vee \overline{l_3}) \wedge (\overline{l_1} \vee \overline{l_3}) \wedge (l_1 \vee \neg x) \wedge (l_2 \vee \neg x) \wedge (l_3 \vee \neg x)$$

Remarque. On a une symétrie sur l_1, l_2 et l_3 dans la formule : une permutation des littéraux dans la formule donne une formule équivalente. On pourra prendre en compte cette symétrie et effectuer, dans les deux questions suivantes, une dissociation de cas sur les valuations en fonction du nombre de littéraux satisfaits par la valuation.

1) Montrer que si $l_1 \vee l_2 \vee l_3$ est valide pour une valuation v sur les variables de l_1, l_2 et l_3, on peut étendre v sur x de façon à satisfaire 7 clauses, mais pas plus.

2) Montrer que si $l_1 \vee l_2 \vee l_3$ n'est pas valide, alors il est impossible d'étendre v sur x en une valuation qui satisfasse plus de 6 de ces clauses.

On va maintenant définir une réduction de 3-SAT vers MAX-2-SAT.

3) Soit φ une formule de 3-SAT avec m clauses. Définir une formule φ' de MAX-2-SAT de taille polynomiale en m et un seuil k tels que φ est satisfiable si et seulement s'il existe une valuation satisfaisant au moins k clauses de φ.

3 Branch and Bound

Lorsque l'on a étudié la satisfiabilité des formules du calcul propositionnel, on a vu que pour une formule du calcul propositionnel à n variables, il existait 2^n valuations possibles des variables.

Dans le cadre de l'algorithme de Quine, on a vu qu'il était possible de représenter la validité ou non des formules du calcul propositionnel pour des valuations données sous une forme arborescente : chaque noeud correspond à une variable, le chemin de gauche ou de droite à partir de ce noeud correspond à la valuation Fausse ou Vraie de cette variable. Si l'on n'effectue pas de simplification des formules, pour une formule à n variables propositionnelles, on obtient un arbre à 2^n feuilles, où la valeur des feuilles est la valeur de vérité de la formule pour la valuation des variables correspondant au chemin dans l'arbre.

On peut, de la même façon, pour une formule sous forme normale conjonctive, construire un arbre similaire et donner comme valeurs aux feuilles un entier, qui est le nombre de clauses **non-satisfaites** par la valuation correspondant au chemin de la feuille.

Trouver une solution optimale au problème MAX-SAT pour une formule revient donc à trouver une feuille de valeur minimale dans l'arbre construit pour cette formule.

Par convention, on considérera dans l'exercice que pour un noeud donné, le fils gauche correspond à la valuation Vraie de la variable associée au noeud, et le fils droit à la valuation Fausse.

4) Construire un tel arbre pour la formule φ_0 à quatre variable x_1, x_2, x_3, x_4 définie ci-dessous :

$$\varphi_0 = (x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_3 \vee x_4) \wedge (x_1 \vee \neg x_4) \wedge (\neg x_1 \vee x_2 \vee x_3)$$
$$\wedge (\neg x_1 \vee \neg x_2) \wedge (\neg x_1 \vee \neg x_3) \wedge (\neg x_2 \vee x_3) \wedge (x_2 \vee \neg x_3)$$

5) Donner une solution optimale pour le problème MAX-SAT pour l'instance φ_0, et le nombre de clauses satisfaites par la valuation.

Faire un parcours exhaustif de l'arbre est une procédure naïve : comme dans le cas de l'algorithme de Quine ou de l'algorithme Min-Max avec élagage $\alpha - \beta$, il est possible d'éviter un

tel parcours exhaustif, en déduisant des informations assurant qu'une solution optimale ne se trouve pas dans le sous-arbre dont on parcourt la racine à un moment donné. La simplification effectuée ici est un cas particulier de la méthode appelée **séparation et évaluation** (Branch and Bound en anglais).

On va dans la suite décrire et implémenter cette méthode en OCaml. Pour cela, on va d'abord représenter les formules du calcul propositionnel sous forme normale conjonctive.

On représentera les littéraux par des entiers : pour des variables $x_1, ..., x_n$, le littéral x_i sera représenté par l'entier i et le littéral $\neg x_i$ par l'entier $-i$.

Les clauses sont représentées par des listes de littéraux et les formules sous forme normale conjonctive par des listes de clauses.

```
type litteral = int ;;
type clause = litteral list ;;
type fnc = clause list ;;
```

6) Définir la variable `phi_0` de type `fnc` représentant la formule φ_0.

7) Définir une fonction `nb_var : fnc -> int` qui prend en entrée une formule et renvoie l'indice maximal n d'une variable propositionnelle x_n dans cette formule (0 pour une formule sans variables).

On représente une valuation sur un ensemble de variables x_i, avec $1 \leq i \leq n$, par un tableau de booléens de taille $n + 1$ (dont on ignore la valeur dans la case d'indice 0) : `v.(i)` vaut `true` ou `false` selon que $v(x_i)$ soit Vrai ou Faux.

8) Définir une variable `v` représentation la valuation (V, F, F, V) pour le quadruplet de variables (x_1, x_2, x_3, x_4).

Dans l'algorithme de séparation et d'évaluation, la séparation correspond à la construction de l'arbre : les sous-arbres dont les racines sont à un niveau donnés correspondent à des solutions partielles, qui partitionnent l'ensemble des solutions.

On cherche une feuille (c'est-à-dire une solution totale) de valeur minimale. Soit m la valeur minimale parmi un ensemble de feuilles déjà parcourues : on souhaite, en parcourant un nouveau noeud interne (c'est-à-dire une solution partielle), borner les valeurs des solutions construites à partir de cette solution partielle. Si on trouve une borne inférieure qui est supérieure à m, on n'a pas besoin de parcourir le sous-arbre car les solutions associées ne seront pas optimales. Si la borne est inférieure à m, il existe potentiellement une solution optimale dans le sous-arbre : on peut parcourir celui-ci. C'est l'étape d'évaluation dans l'algorithme.

Pour la question suivante, on représente une valuation partielle des variables $x_1, ..., x_n$ jusqu'à une variable x_k par un tableau de booléens dont les valeurs jusqu'à l'indice k correspondent aux valuations des variables x_1 à x_k, et les valeurs aux indices supérieurs sont arbitraires.

9) Ecrire une fonction `insat_clause : bool array -> int -> clause -> bool` qui prend en entrée une valuation partielle `v` jusqu'à une variable x_k, l'entier `k` correspondant, une formule `phi` et renvoie `true` ou `false` selon que la valuation partielle `v` puisse on non êtr complétée en valuation satisfaisant `phi`.

Remarque. On pourra utiliser la fonction `List.for_all`, telle que pour une fonction `f` de type `'a -> bool` et une liste `[a1;...;an]` de type `'a list`, `List.for_all f [a1;...;an]`

renvoie la valeur du booléne `f a1 && f a2 && ... && f an`.

10) En déduire une fonction `insat : bool array -> int -> fnc -> int` qui prend en entrée une valuation partielle `v` jusqu'à une variable x_k, un entier `k`, une formule `phi` et renvoie le nombre de clauses de `phi` qui ne peuvent pas être satisfaites par une valuation étendant `v`.

Dans la méthode par séparation et évaluation, on commence par initialiser le minimum en calculant le nombre de clauses non-satisfaites pour la valuation correspondant à la branche la plus à gauche dans l'arbre (c'est-à-dire la valuation (V, V, V, V) pour (x_1, x_2, x_3, x_4) dans le cas de φ_0).

Ensuite, on effectue un parcours partiel de l'arbre. La fonction `insat` permet de donner une borne inférieure au nombre de clauses non-satisfiables à partir d'une valuation partielle pour les variables d'une formule :

- Si cette borne inférieure est supérieure ou égale au minimum connu à un instant donné, on ne fait pas le parcours du sous-arbre, dans lequel on ne trouvera pas de solution améliorant le minimum.
- Sinon, on parcourt le sous-arbre en parcourant ses deux fils et en appliquant la même procédure. Si on atteint une feuille dont la valeur est inférieure strictement au minimum actuel, on met à jour le minimum et la valuation associée (c'est-à-dire la valuation correspondant au chemin de la feuille).

11) Effectuer la procédure à la main pour φ_0, en indiquant à côté de chaque noeud le nombre de clauses insatisfiables pour la valuation partielle correspondante, et en barrant les branches parentes des sous-arbres non-parcourues.

12) Ecrire une fonction `maxSat : fnc -> bool array` prend en entrée une formule sous forme normale conjonctive et renvoie une valuation maximisant le nombre de clauses satisfaites par la formule en appliquant la procédure de séparation et évaluation.