TD 5 : Méthode de Monte-Carlo

Exercice 1. *Manipulations de formules booléennes*

On souhaite représenter et manipuler des formules DNF. On adopte la convention suivante : une clause $\bigwedge_i \ell_i$ est représentée par une liste d'entiers, l'entier k représentant le littéral x_k s'il est positif et $\neg x_{-k}$ s'il est négatif ; une formule ϕ est représentée par une liste de clauses. La taille d'une formule est la somme des listes qui la composent. On représente une affectation par un tableau A, tel que $A[i]$ vaut 1 si x_i est TRUE et -1 sinon.

1. Donner la représentation de la formule $(x_1 \wedge \neg x_2 \wedge x_3) \vee (x_2 \wedge x_4) \vee (\neg x_1 \wedge x_2)$ et compter son nombre de modèles.
2. Écrire un algorithme qui prend en entrée une formule et une affectation A et renvoie l'évaluation de la formule sur l'affectation. Analyser sa complexité.
3. Écrire un algorithme qui teste si une formule est satisfiable et renvoie une affectation satisfaisante si tel est le cas. Analyser sa complexité.
4. Écrire un algorithme qui étant donnée une formule, renvoie la formule obtenue en supprimant toutes les clauses insatisfiables. Analyser sa complexité.

Exercice 2. *Amélioration de l'algorithme* MONTE-CARLO-DNF

L'étape la plus coûteuse de l'algorithme MONTE-CARLO-DNF est le test $(j,a) \in A^\star_\phi$ puisqu'on doit tester, dans le pire des cas ($j = m$), si a satisfait chaque clause de la formule. On étudie dans cet exercice une variante qui permet d'améliorer la complexité globale de l'algorithme. Soit ϕ une formule DNF à n variables et m clauses. Pour chaque modèle $a \in A_\phi$, on note $c_a \geq 1$ le nombre de clauses de ϕ qu'il satisfait. On considère l'algorithme suivant :

```
ClauseAléatoire(a, φ):
    t ← 0
    Répéter :
        Tirer j ∈ {1, ..., m} aléatoirement
        t ← t + 1
    tant que a ∉ A_j
    Renvoyer t/m
```

1. Montrer que l'espérance de la valeur renvoyée par ClauseAléatoire est $1/c_a$. Quelle est l'espérance de son temps de calcul ?

L'idée de l'amélioration est de remplacer le test $(j,a) \in A^\star_\phi$ par un appel à ClauseAléatoire(a,ϕ) : au lieu d'incrémenter le compteur de 1 si $(j,a) \in A^\star_\phi$, on l'incrémente à tous les coups de la valeur renvoyée par ClauseAléatoire.

2. Montrer que l'espérance de la valeur renvoyée par l'algorithme reste la même. *Indication. On peut montrer que l'espérance de l'incrément à chaque tirage est la même avec le nouvel algorithme qu'avec l'ancien.*

Malheureusement, comme l'espérance du temps de calcul de ClauseAléatoire est égale au temps d'une itération de l'algorithme d'origine on ne gagne rien (pour l'instant).
On note $\mu = |A_\phi|/|A^\star| \geq 1/m$.

3. Montrer qu'en effectuant $k(\mu) = 3\ln(2/\epsilon)/\mu\delta^2$ tirages dans l'algorithme MONTE-CARLO-DNF, la valeur renvoyée par l'algorithme est dans l'intervalle $(1 \pm \delta)|A_\phi|$ avec probabilité $\geq 1 - \epsilon$.
4. Montrer qu'en utilisant $k = k(\mu)$ dans le nouvel algorithme, l'espérance du temps de calcul total est $O(mn\log(1/\epsilon)/\delta^2)$.

Pour *simuler* le fait de faire $k(\mu)$ tirages, on utilise la technique suivante. On pose $T = cm\ln(2/\epsilon)/\delta^2$ pour une certaine constante c. On applique l'algorithme en comptant le nombre total de tests $a \notin A_j$ effectués dans les différents appels à ClauseAléatoire. Dès que ce nombre dépasse T, on s'arrête et on renvoie $T|A^\star|/(mK_T)$ où K_T est le nombre d'appels à ClauseAléatoire effectués (et terminés) pendant l'algorithme.

5. Montrer que cet algorithme a bien complexité $O(mn\log(1/\epsilon)/\delta^2)$.
6. *(difficile)* Montrer qu'il renvoie le *bon* résultat pour une certaine valeur de c.