COMPLEX - TD 09
décembre 2023

Énoncé

Exercice 1 : Coupe maximum

En théorie des graphes, une *coupe* d'un graphe est une partition des sommets en deux sous-ensembles (on appelle parfois aussi coupe l'ensemble des arêtes ayant une extrêmité dans chaque sous-ensemble de la partition).

Étant donné un graphe non orienté $G = (V, E)$, un sous-ensemble S de V définit la coupe $\{S, V \setminus S\}$ et le *poids* de cette coupe, noté $c(S)$ est le nombre d'arêtes de E ayant une extrêmité à l'intérieur de cet ensemble et l'autre à l'extérieur :

$$c(S) = \#\{\{i, j\} \in E \mid i \in S, j \in V \setminus S\}.$$

Le problème de la coupe maximum est le suivant : étant donné un graphe non orienté $G = (V, E)$, trouver un coupe (ou un sous-ensemble S) de poids maximum (parmi toutes les coupes de G). Il s'agit d'un problème NP-difficile. Nous allons étudier dans cet exercice un algorithme probabiliste de type Monte-Carlo efficace qui retourne une 4-approximation de la coupe maximum (*i.e.* dont le poids est au plus 4 fois plus petit que celui d'une coupe maximum).

1.a] Soit $e \in E$ une arrète du graphe. Montrer que pour un ensemble S aléatoire, l'arête e a une extrêmité à l'intérieur de S et l'autre à l'extérieur avec probabilité $1/2$.

1.b] En déduire que, pour un ensemble S aléatoire, le poids moyen de la coupe définie par S est égale à $(\#E)/2$.

1.c] Considérons l'algorithme suivant : l'ensemble S est initialement vide et pour chaque nœud $v \in V$, le nœud v est ajouté à l'ensemble S avec probabilité $1/2$. Lorsque tous les nœuds ont été considérés, l'algorithme retourne l'ensemble S construit (qui définit la coupe $\{S, V \setminus S\}$).

Montrer que cet algorithme retourne une 4-approximation de la coupe maximum avec probabilité au moins $1/3$.

Indication : On pourra appliquer l'inégalité de Markov au nombre d'arêtes ne traversant pas la coupe construite.

1.d] Expliquer comment utiliser l'algorithme de la question précédente pour avoir un algorithme qui retourne une 4-approximation de la coupe maximum avec probabilité au moins $1 - (2/3)^k$ pour tout entier $k \in \mathbb{N}$. Donner sa complexité en temps et en espace.

Exercice 2 : MAX-SAT

Le problème MAX-SAT consiste, étant donnée une formule logique propositionnelle donnée sous forme d'une conjonction de clauses (CNF dans la suite) à trouver le nombre maximum de clauses que l'on peut satisfaire simultanément avec une assignation des variables. En cours, nous avons étudié le cas particulier de MAX-3SAT qui considère le problème uniquement pour les CNF conjonctions de 3-clauses.

Rappel : Un littéral ℓ_i est une variable propositionnelle x_j (littéral positif) ou la négation d'une variable propositionnelle $\neg x_j$ (littéral négatif). Une k-clause (pour $k \in \mathbb{N} \setminus \{0\}$) est une disjonction de la forme $\ell_1 \vee \ell_2 \vee \cdots \vee \ell_k$ où ℓ_i est un littéral pour $i \in \{1, \ldots, k\}$. Dans la suite, nous supposerons toujours que les variables apparaissant dans k littéreaux d'une k-clauses sont toutes distinctes.

2.a] Soient $k \in \mathbb{N} \setminus \{0\}$ et C une k-clause. Montrer que la probabilité que C soit satisfaite pour un choix uniformément aléatoire d'une assignation dans $\{0,1\}^k$ est égale à $1 - 2^{-k}$.

2.b] En déduire que pour une CNF conjonction de n clauses (de longueur arbitraire), un choix uniformément aléatoire d'une assignation satisfait en moyenne au moins $n/2$ clauses.

2.c] Proposer un algorithme probabiliste polynomial de type Las Vegas qui étant donnée une CNF conjonction de n clauses retourne une assignation qui satisfait au moins $(n/2 - 1)$ clauses. Préciser et justifier son temps d'exécution moyen.

2.d] Cet algorithme est moins performant que l'algorithme vu en cours pour résoudre MAX-3SAT en raison de présences possible de k-clauses avec $k \in \{1, 2\}$.
Donner une CNF conjonction de n clauses à un littéral (i.e. de 1-clauses) pour laquelle il n'existe pas d'assignation qui satisfait strictement plus de $n/2$ clauses.

2.e] Soit $k \in \mathbb{N} \setminus \{0\}$ et soit $p \in [1/2, 1]$ un nombre réel. Soit C une k-clause qui n'est pas une 1-clause avec un littéral négatif (*i.e.* $k > 1$ ou C est une 1-clause de la forme $C = x_i$). Nous considérons un choix aléatoire (non-uniforme) d'une assignation où chaque variable prend la valeur 1 avec probabilité p et la valeur 0 avec probabilité $1 - p$.
Montrer que la probabilité que C soit satisfaite pour cette distribution des assignations est supérieure ou égale à $\min(p, 1 - p^2)$.

Indication : On pourra distinguer les 1-clauses et les k-clauses avec $k > 1$ et utiliser sans démonstration le fait que pour $p \in [1/2, 1]$ et $a, b \in \mathbb{N}$ avec $a + b \geq 2$, nous avons $1 - p^a(1-p)^b \geq 1 - p^2$.

2.f] Calculer la valeur γ de p qui maximise cette probabilité.

2.g] Décrire brièvement (en utilisant les questions précédentes) un algorithme probabiliste polynomial de type Las Vegas qui étant donnée une CNF conjonction de n clauses (mais aucune 1-clause avec un littéral négatif) retourne une assignation des variables qui satisfait au moins $(\gamma \cdot n - 1)$ clauses.

Compléments

Exercice 3 : Algorithme probabiliste pour **3-SAT**

Nous considérons l'algorithme suivant :
Rappel : Un littéral ℓ_i est une variable propositionnelle x_j (littéral positif) ou la négation d'une variable propositionnelle $\neg x_j$ (littéral négatif). Une k-clause (pour $k \in \mathbb{N} \setminus \{0\}$) est une disjonction de la forme $\ell_1 \vee \ell_2 \vee \cdots \vee \ell_k$ où ℓ_i est un littéral pour $i \in \{1, \ldots, k\}$. Dans la suite, nous supposerons toujours que les variables apparaissant dans k littéreaux d'une k-clauses sont toutes distinctes.

Algorithme 1 : Algorithme probabiliste pour 3-SAT

Entrée : Formule Φ de m clauses $C_1, \ldots, C_m$ en n variables $x_1, \ldots, x_n$; $T \in \mathbb{N}$
$\triangleright$ `$C_1, \ldots, C_k$ pour $k \leq m$ sont les 3-clauses de $\Phi$`
$\triangleright$ `$C_{k+1}, \ldots, C_m$ sont les 1-clauses et les 2-clauses de $\Phi$`

Sortie : $b \in \{\text{SATISFIABLE}, \text{NON-SATISFIABLE}\}$

1 **pour** *j de 1 à T* **faire**
2     $\Phi' \leftarrow \{C_{k+1}, \ldots, C_m\}$ $\triangleright$ Φ' est constituée des 1-clauses et de 2-clauses de Φ
3   **pour** *i de 1 à k* **faire**
4         $\ell_1^i \vee \ell_2^i \vee \ell_3^i \leftarrow C_i$
5         $t \longleftarrow \{1, 2, 3\}$ $\triangleright$ supprime un littéral aléatoire à la 3-clause C_i
6   **si** $t = 1$ **alors**
7             $C_i' \leftarrow \ell_2^i \vee \ell_3^i$
8   **si** $t = 2$ **alors**
9             $C_i' \leftarrow \ell_1^i \vee \ell_3^i$
10   **si** $t = 3$ **alors**
11             $C_i' \leftarrow \ell_1^i \vee \ell_2^i$
12         $\Phi' \leftarrow \Phi' \cup \{C_i'\}$ $\triangleright$ ajoute la 2-clause C_i' obtenue à Φ'
13   **si** Φ' *est satisfiable* **alors**
14   **retourner** SATISFIABLE $\triangleright$ en temps $O(n+m)$ car Φ' est une formule 2-SAT
15 **retourner** NON-SATISFIABLE

Le problème 3-SAT consiste, étant donné une formule logique propositionnelle Φ donnée sous forme d'une conjonction de k-clauses avec $k \leq 3$, à décider si il existe une assignation des variables satisfaisant toutes les clauses.

3.a] Montrer que si cet algorithme retourne SATISFIABLE sur une formule booléenne Φ alors la formule Φ est effectivement satisfiable.

Nous allons désormais analyser la probabilité que l'algorithme retourne NON-SATISFIABLE sur une formule booléenne Φ satisfiable. Soit $\vec{z} \in \{0,1\}^n$ une assignation des variables qui rend la formule Φ vraie.

3.b] Montrer que pour tout $i \in \{1, \ldots, k\}$ (dans la boucle des lignes 3 à 12 de l'algorithme), l'assignation $\vec{z}$ rend C_i' vraie avec probabilité supérieure ou égale à $2/3$.

3.c] En déduire que l'assignation $\vec{z}$ rend la formule Φ' (construite à la fin de la boucle des lignes 3 à 12) vraie avec probabilité supérieure ou égale à $(2/3)^k$.

3.d] Donner un choix du paramètre T (en fonction de n, m, k) qui rend la probabilité d'erreur de cet algorithme inférieure à $1/e$. Une analyse précise de cette probabilité d'erreur et du temps d'exécution de l'algorithme obtenu est demandée.

Algorithme 2 : Algorithme de Hirsch pour MAX-3-SAT

Entrée : Formule Φ de m clauses $C_1, \ldots, C_m$ en n variables $x_1, \ldots, x_n$; $T \in \mathbb{N}$

Sortie : $(y, \mu) \in \{0,1\}^n \times \{0, \ldots, m\}$ tel que $\Phi(y)$ satisfait μ clauses

1 $\mu^\star \leftarrow -1$; $y^\star \leftarrow (0, \ldots, 0)$ ▷ `solution optimale`
2 **pour** j *de* 1 *à* T **faire**
3     $y = (y_1, \ldots, y_n) \xleftarrow{} \{0,1\}^n$
4     $\mu \leftarrow \#\{i \in \{1, \ldots, m\} | C_i(y) = 1\}$ ▷ `nombre de clauses satisfaites par` y
5   **si** $\mu > \mu^\star$ **alors**
6         $\mu^\star \leftarrow \mu$; $y^\star \leftarrow y$ ▷ `mise à jour de la solution optimale`
7   **pour** i *de* 1 *à* n **faire**
8   **si** $\Phi(y) = 0$ **alors**
9   considérer une clause C_t (arbitraire) non satisfaite de Φ
10             $j \xleftarrow{} \textsc{Variables}(C_t)$ ▷ `tirage d'une variable dans` C_t
11             $y_j \leftarrow 1 - y_j$ ▷ `changement de la valeur de` y_j
12             $\mu \leftarrow \#\{i \in \{1, \ldots, m\} | C_i(y) = 1\}$ ▷ `nombre de clauses satisfaites par` y
13   **si** $\mu > \mu^\star$ **alors**
14                 $\mu^\star \leftarrow \mu$; $y^\star \leftarrow y$ ▷ `mise à jour de la solution optimale`
15   **sinon**
16   **retourner** $(y^\star, \mu^\star)$
17 **retourner** $(y^\star, \mu^\star)$

Exercice 4 : Algorithme de Hirsch pour MAX-3-SAT

Dans cet exercice, nous considérons un algorithme probabiliste du à E. A. Hirsch pour résoudre le problème MAX-3-SAT de façon approchée. L'algorithme est inspiré de l'algorithme de Schöning pour le problème 3-SAT vu en cours.

Soit Φ une formule booléenne en forme normale conjonctive formée de m clauses contenant toutes au plus 3 littéraux. Soit $\ell \leq m$ le nombre maximal de clauses simultanément satisfiables pour Φ et soit $z = (z_1, \ldots, z_n)$ une assignation (optimale) des variables telle que $\Phi(z_1, \ldots, z_n)$ satisfait ℓ clauses.

4.a] Montrer que $\ell \geq m/2$.

Soit $\epsilon > 0$. Nous disons qu'une assignation des n variables de Φ est *admissible* si elle satisfait au moins $(1-\varepsilon)\ell$ clauses. Notons que si lors de l'exécution de l'algorithme, nous trouvons une assignation des n variables qui est admissible, alors l'algorithme retourne une assignation admissible.

4.b] Considérons une assignation qui ne satisfait pas strictement plus de $m - (1-\varepsilon)\ell$ clauses de Φ dans une itération de la boucle **pour** … **faire** des lignes 7 à 16. Notons probabilité p_ε que l'algorithme change la valeur d'une variable qui a une valeur différente dans cette assignation et dans z. Montrer que

$$p_\varepsilon \geq \frac{\varepsilon\ell}{3(m-(1-\varepsilon)\ell)}$$

4.c] En déduire que

$$p_\varepsilon \geq \frac{\varepsilon}{3(1+\varepsilon)}$$

4.d] Supposons que, lors de l'initialisation de la boucle **pour** ... **faire** des lignes 2 à 16, l'assignation y diffère de celle de z en exactement t variables (ce qui se produit avec probabilité $\binom{n}{t}/2^{-n}$). Montrer que pour une telle initialisation, l'algorithme obtient une assignation admissible avec probabilité supérieure ou égale à p_ε^t.

4.e] En sommant pour toutes les valeurs possibles de t, montrer que lors de chaque itération de la boucle **pour** ... **faire** des lignes 2 à 16, l'algorithme obtient une assignation admissible avec probabilité supérieure ou égale à $q = \left(\frac{1}{2}\left(1+\frac{\varepsilon}{3(1+\varepsilon)}\right)\right)^n$.

4.f] Montrer que pour $T = 1/q$, l'algorithme obtient une assignation admissible avec probabilité supérieure ou égale à $\exp(-1) \simeq 0.368$ (et dans ce cas $T = c^n$ avec $c = (2-2\varepsilon/(3+4\varepsilon)) < 2$).