IFT800 - Série d'exercices #2

Manuel Lafond

Exercice 1. Considérez la variante suivante de MAX-CUT (le w est pour *weighted*).

w-MAX-CUT
Entrée : graphe $G = (V, E)$ et poids $w : E \to \mathbb{R}^{\geq 0}$ sur les arêtes.
Sortie : une bipartition (V_1, V_2) qui maximise $\sum_{uv \in E(V_1, V_2)} w(uv)$.

Considérez l'algorithme de recherche locale qui démarre d'une solution arbitraire (V_1, V_2), et tant qu'il y a un sommet v dans V_1 ou dans V_2 tel que déplacer v améliore la valeur de la bipartition, on le fait.

Montrez que ceci retourne une solution telle que $APP \geq 1/2 \cdot OPT$.
Suggestion. Faites comme dans la version sans poids, mais montrez que chaque sommet a la moitié des poids des ses arêtes qui traversent de l'autre côté.

Défi. Montrez toutefois que cet algorithme ne termine pas toujours en temps polynomial. Ce n'est donc pas techniquement une 1/2-approximation. Ceci est probablement difficile : je n'ai pas essayé — je sais seulement que j'ai vu le résultat quelque part.

Exercice 2. Il existe une 1/2-approximation de type glouton au problème MAX-CUT. L'idée est de construire notre bipartition (V_1, V_2) en ajoutant un sommet à la fois. Pour chaque v, on ajoute v dans la partie où il a le moins de voisins. On dénote par $N(v)$ l'ensemble des voisins de v dans G.

```
fonction maxCutGlouton(U, S, w)
    Soit xy ∈ E une arête arbitraire
    V_1 = {x}
    V_2 = {y}
    R = V \ {x, y}
    tant que R ≠ ∅ faire
        Soit v ∈ R
        si |N(v) ∩ V_1| > |N(v) ∩ V_2| alors
            V_2.append(v)
        sinon
            V_1.append(v)
        fin
        R.remove(v)
    fin
```

Montrez que cet algorithme est une 1/2-approximation.
Suggestion: démontrez par induction qu'après chaque ajout d'un sommet v, si on ne consdière que les sommets ajoutés jusqu'à maintenant, le nombre d'arêtes qui traversent V_1 et V_2 est au moins la moitié du nombre d'arêtes à l'intérieur de $V_1 \cup V_2$.

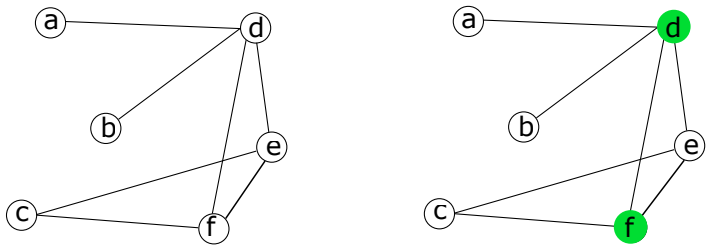
Exercice 3. Montrez que l'algorithme de la question précédente pour MAX-CUT n'est pas une $(1/2 + \epsilon)$-approximation pour tout $\epsilon > 0$, et donc que l'analyse est serrée.
Indice. Difficile de donner un bon indice. C'est le genre de question qui est facile, mais seulement une fois qu'on connaît la réponse. Disons qu'à part x les y, les autres sommets n'ont pas beaucoup de voisins.

Exercice 4. Dans le problème Δ-MIN-COLORING, on a un graphe $G = (V, E)$ dans lequel chaque sommet a au plus Δ voisins. On veut colorier les sommets de G avec un minimum de couleurs de façon à ce que si $uv \in E$, alors u et v reçoivent deux couleurs différentes. On peut supposer que les couleurs sont des entiers positifs.

L'approche gloutonne consiste à itérer à travers chaque sommet dans un ordre quelconque. À chaque sommet v, on attribue à v comme couleur le plus petit entier qui n'est pas déjà utilisé par un de ses voisins.

Montrez que l'approche gloutonne est une $(\Delta + 1)$-approximation.
Suggestion. Combien de couleurs faudra-t-il au maximum? Une fois que vous savez, une borne triviale sur OPT peut s'appliquer.

Exercice 5. Soit $G = (V, E)$ un graphe. Rappelons qu'un ensemble $X \subseteq V$ est *dominant* si, pour tout $v \in V \setminus X$, v a au moins un voisin dans X. Dans l'exemple ci-dessous, $\{d, f\}$ est dominant car chacun de $\{a, b, c, e\}$ a au moins un voisin dans $\{d, f\}$.



On cherche un ensemble dominant de taille minimum.

MIN-DOMSET
Entrée : un graphe $G = (V, E)$;
Sortie : un ensemble dominant $X \subseteq V$ qui minimise $|X|$.

Donnez le pseudo-code d'une $O(\log n)$-approximation. Vous devez ensuite justifier que votre algorithme est bel et bien une H_n-approximation.

Suggestion. Transformez une instance de MIN-DOMSET en une instance de SET-COVER. De cette façon, vous n'avez pas de démonstration profonde à faire.

Exercices difficiles

Ces exercices sont encore plus optionnels que les précédents, car ils posent de bons défis et sont surtout ici pour ceux et celles qui veulent approfondir.

Exercice 6. Considérez la version de SET-COVER avec des poids.

> w-SET-COVER
> **Entrée :** univers U, ensembles $S = \{S_1, \ldots, S_m\}$ et poids $w : S \to \mathbb{R}^{\geq 0}$.
> **Sortie :** des ensembles $S^* \subseteq S$ qui couvrent U qui minimisent $\sum_{S' \in S^*} w(S')$.

Le choix glouton correspond à ajouter à S^* l'ensemble S_i qui minimise le coût par nouvel élément couvert $\frac{w(S_i)}{|S_i \cap R|}$. L'intuition est de prendre le S_i qui couvre les éléments de la façon la plus "efficace".

Considérez l'algorithme suivant.

fonction *setCoverGloutonW(U, S, w)*
    $R = U$ //Éléments restant à couvrir
    tant que $R \neq \emptyset$ **faire**
        Soit $S_i \in S$ qui minimise $\frac{w(S_i)}{|S_i \cap R|}$ `// Choix glouton`
        $S^*.append(S_i)$
        pour $u_k \in S_i \cap R$ **faire**
            //ceci n'est pas requis par l'algorithme. C'est seulement
            //pour définir un coût à utiliser pour l'analyse
            définir $cout(u_k) = \frac{w(S_i)}{|S_i \cap R|}$
        fin
        $R = R \setminus S_i$
        Enlever de S les ensembles S_i tels que $S_i \cap R = \emptyset$
    fin
    return S^*

Montrez que ceci est une H_n-approximation, où $H_n = 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{n}$.

Suggestion: Définissez $cout(u_k) = \frac{w(S_i)}{|S_i \cap R|}$ comme dans l'algorithme, où S_i est l'ensemble qui a couvert u_k et R les éléments restants juste avant de couvrir u_k. Commencez par montrer que la somme des coûts est égale à APP (ceci est parce que chaque ensemble S_i choisi distribue son poids dans les coûts des éléments qu'il couvre).

Une idée importante est que avant de couvrir le k-ième élément u_k, on avait $|R| \geq n - k + 1$. On est capable de couvrir ces R éléments avec un coût total de OPT. La partie difficile (selon moi) est de démontrer qu'il existe toujours S_i tel que $\frac{w(S_i)}{|S_i \cap R|} \leq \frac{OPT}{|R|}$. Pour avoir tous vos points (donc 0/0), vous pouvez le démontrer. Mais sinon, vous pouvez supposer que c'est le cas et compléter votre preuve, ce qui démontre déjà une excellente compréhension.

L'implication de $\frac{w(S_i)}{|S_i \cap R|} \leq \frac{OPT}{|R|}$ est que, combiné à $|R| \geq n - k + 1$, on a

$$cout(u_k) = \frac{w(S_i)}{|S_i \cap R|} \leq \frac{OPT}{n - k + 1}$$

Prenez ensuite la somme des coûts pour comparer APP et OPT.

Exercice 7. Considérez le problème du k-SET-COVER, dans lequel chaque ensemble contient exactement k éléments.

Démontrez que l'algorithme glouton tel que vu en classe est une H_k-approximation. *NOTE:* cette question n'est vraiment pas facile (du moins selon moi). Voyez-la comme un défi optionnel. Essayez de simplifier en fixant $k = 3$. L'analyse telle ci-haut ne semble pas fonctionner, il faut changer notre perspective sur l'analyse. Attribuez un cout de $1/|S \cap R|$ aux éléments à couvrir comme dans la question précédente. Regardez ensuite, pour chaque ensemble S_i d'une solution optimale, le coût qu'il a fallu payer pour chacun de ses éléments dans l'algorithme glouton.