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.

Solution.

On observe qu'à la fin de l'exécution de l'algorithme, chaque $v \in V_1$ a la moitié des poids de ses arêtes avoisinantes qui traverse à V_2. C'est-à-dire, pour tout $v \in V_1$,

$$\sum_{z \in N(v) \cap V_2} w(vz) \geq \frac{1}{2} \sum_{z \in N(v)} w(vz)$$

Ceci est parce que sinon, l'algorithme aurait transféré v vers V_2 et aurait obtenu une meilleure bipartition. De la même façon, chaque $v \in V_2$ a la moitié de ses

poids vers V_1. On note aussi que APP est égal au poids des arêtes de V_1 vers V_2, plus le poids des arêtes de V_2 vers V_1, divisé par 2 car ceci compte chaque arête deux fois. On obtient donc

$$\begin{aligned} APP &\geq \frac{1}{2}\left(\frac{1}{2}\sum_{v\in V_1}\sum_{z\in N(v)} w(vz) + \frac{1}{2}\sum_{v\in V_2}\sum_{z\in N(v)} w(vz)\right) \\ &= \frac{1}{4}\sum_{v\in V}\sum_{z\in N(v)} w(vz) \end{aligned}$$

La meilleure solution possible est celle où chaque arête traverse, et donc que tous les voisinages sont de l'autre côté. On a donc

$$OPT \leq \frac{1}{2}\sum_{v\in V}\sum_{z\in N(v)} w(vz)$$

et on voit qu'on a une 1/2-approximation.

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
    V1 = {x}
    V2 = {y}
    R = V \ {x, y}
    tant que R ≠ ∅ faire
        Soit v ∈ R
        si |N(v) ∩ V1| > |N(v) ∩ V2| alors
            V2.append(v)
        sinon
            V1.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 $V_1 \cup V_2$.

Solution.

On va démontrer qu'après chaque itération, la moitié des arêtes dont les deux sommets sont dans $V_1 \cup V_2$ traversent V_1 et V_2. On procède par induction sur le nombre d'itérations effectuées, qu'on va dénoter i. Comme cas de base, quand $i = 0$ (donc avant d'entreprendre la 1re itération), on a $V_1 = \{x\}$ et $V_2 = \{y\}$ et ici, toutes les arêtes possibles traversent.

Pour l'induction, on suppose que la moitié des arêtes dans $V_1 \cup V_2$ traversent après l'itération $i - 1$. Soit m^{tr} le nombre de telles arêtes qui traversent après l'itération $i - 1$, et soit m^* le nombre d'arêtes dont les deux bouts sont dans $V_1 \cup V_2$ toujours après l'itération $i - 1$. Donc par induction, on a $m^{tr} \geq m^*/2$.

Considérons le sommet v ajouté à l'itération i. Après l'itération i, on a donc

introduit k nouvelles arêtes à l'intérieur de $V_1 \cup V_2$, où $k = |N(v) \cap (V_1 \cup V_2)|$. Après l'itération, on a donc $m^* + k$ arêtes dans $V_1 \cup V_2$. Supposons que v est ajouté à V_1. Alors v doit avoir au moins $k/2$ voisins dans V_2, car sinon on aurait ajouté v à V_1. Le nombre d'arêtes traversantes après l'itération i est donc au moins

$$m^{tr} + k/2 \geq m^*/2 + k/2 = (m^* + k)/2$$

ce qui prouve qu'au mois la moitié des arêtes traversent après l'itération i. Si v est ajouté à V_2, la preuve est identique.

Puisque notre énoncé est vrai pour chaque itération, à la toute fin, on aura au moins la moitié du total des arêtes qui traversent. C'est-à-dire, $APP \geq |E|/2 \geq OPT/2$.

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.

Solution.

On va construire un exemple. Démarrez avec deux sommets x, y et ajoutez l'arête xy. Ensuite, ajoutez des sommets $v_1, v_2, \ldots, v_n$ tel que $N(v_i) = \{x, y\}$ pour chaque v_i.

Il n'est pas trop difficile de voir que $OPT = 2n$ avec la bipartition $V_1 = \{x, y\}$ et $V_2 = \{v_1, \ldots, v_n\}$ (on a $2n$ car chaque v_i amène deux arêtes traversantes). Si l'algorithme choisit xy comme départ, chaque fois qu'on ajoute un v_i, il y aura seulement une des arêtes touchant v_i qui est traversante. Le nombre d'arêtes traversantes de l'algorithme sera donc $APP = n + 1$ (le $+1$ pour compter xy).

On cherche le c tel que $APP = c \cdot OPT$. On a $APP = n+1 = c \cdot 2n$, et donc $c = \frac{n+1}{2n} = \frac{1}{2} + \frac{1}{n}$. En faisant tendre n vers l'infini, on voit que $APP \geq 1/2 + \epsilon$ est impossible pour tout $\epsilon > 0$.

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.

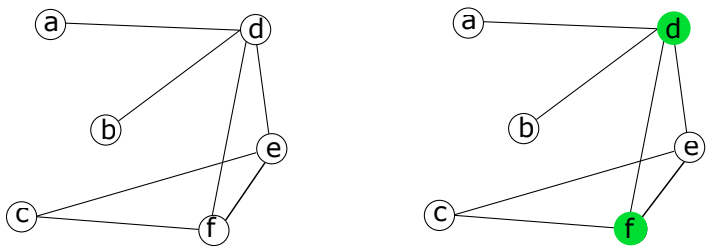
Solution.

On peut argumenter que l'algorithme n'utilisera jamais plus que $\Delta + 1$ couleurs. Supposons qu'on colorie les sommets dans l'ordre $v_1, \ldots, v_n$ avec l'approche gloutonne. Quand on colorie v_i, au pire, chacun de ses Δ voisins a déjà colorié, chacun avec une couleur différente. Il reste quand même une couleur disponible parmi les $\Delta + 1$ pour colorier v_i, et donc $\Delta + 1$ couleurs suffisent.

Pour argumenter que c'est une $(\Delta + 1)$-approximation, il suffit d'observer que $OPT \geq 1$. On a

$$APP \leq \Delta + 1 \leq (\Delta + 1) \cdot 1 \leq (\Delta + 1) \cdot OPT$$

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.

Solution.

Soit G une instance de MIN-DOMSET. On transforme en une instance (U, S) équivalente de SET-COVER, où U est l'univers et S les ensembles.

On pose $U = V(G)$ et, pour tout $v \in V(G)$, on ajoute à S l'ensemble $S_v = N(v) \cup \{v\}$. L'idée est qu'un ensemble dominant est comme SET-COVER, car l'univers à couvrir est $V(G)$ et chaque v couvre S_v.

Pour obtenir notre $O(\log n)$-approximation, il suffit d'exécuter l'algorithme glouton pour SET-COVER. Si l'algorithme retourne $S' \subseteq S$ pour SET-COVER, on retourne ensuite $\{v \in V(G) : S_v \in S'\}$, qui est un ensemble dominant.

Soit APP_{dom} la valeur de la solution retournée ci-haut pour l'ensemble dominant, et APP_{cov} pour SET-COVER. On a $APP_{dom} = APP_{cov}$. On définit OPT_{dom} et OPT_{cov} de la même façon. Il reste à montrer que $APP_{dom} \in O(\log n)OPT_{dom}$. On remarque que si X est un ensemble dominant optimal de G, alors $\{S_v : v \in X\}$ est couvre U. Conversément, si S' est un ensemble couvrant optimal, alors $\{v \in V(G) : S_v \in S'\}$ est dominant. Ceci implique que $OPT_{dom} = OPT_{cov}$. On obtient

$$APP_{dom} = APP_{cov} \leq c \log n OPT_{cov} = c \log OPT_{dom}$$

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.

Solution.

On remarque d'abord que

$$\sum_{u_k} cout(u_k) = \sum_{S' \in S^*} w(S')$$

parce que quand un S_i est ajouté à S^*, il distribue un total de $w(S_i)$ à travers les éléments $R \cap S_i$ qu'il couvre.

En supposant que les u_k sont triés par ordre de couverture (comme vu en classe), on peut argumenter que $\frac{OPT}{n-k+1} \geq cout(u_k)$ comme dans la version sans les poids.

On va le démontrer plus bas. Si on suppose que c'est le cas, on obtient

$$APP = \sum_{S_i \in S^*} w(S_i) = \sum_{k=1}^{n} cout(u_k) \leq \sum_{k=1}^{n} \frac{OPT}{n - k + 1} = H_n \cdot OPT$$

Il nous reste à démontrer que $\frac{OPT}{n-k+1} \geq cout(u_k)$ pour tout k. Soit R l'ensemble d'éléments restant juste avant de couvrir u_k. On a $|R| \geq n - k + 1$. On sait qu'on peut couvrir R avec un coût total de OPT. On va montrer qu'il existe $S_i \in S$ tel que $\frac{w(S_i)}{|S_i \cap R|} \leq \frac{OPT}{|R|}$.

Supposons que ce n'est pas le cas. Soit $\hat{S}$ une solution optimale. Alors,

pour tout $S_i \in \hat{S}$ dans la solution optimale, on aurait $\frac{w(S_i)}{|S_i \cap R|} > \frac{OPT}{|R|}$, ce qui implique

$$w(S_i) > \frac{OPT \cdot |S_i \cap R|}{|R|}$$

et donc

$$\sum_{S_i \in \hat{S}} w(S_i) > \sum_{S_i \in \hat{S}} \frac{OPT \cdot |S_i \cap R|}{|R|} = \frac{OPT}{|R|} \cdot \sum_{S' \in \hat{S}} |S_i \cap R| \geq \frac{OPT}{|R|} \cdot |R|$$

(pour la dernière inégalité, puisque $\hat{S}$ couvre R, la somme des $|S_i \cap R|$ est au moins $|R|$)

On obtient donc $\sum_{S_i \in \hat{S}} w(S_i) > OPT$, une contradiction parce que $\hat{S}$ est une solution optimale de coût OPT.

Donc, il existe bel et bien S_i tel que $\frac{w(S_i)}{|S_i \cap R|} \leq \frac{OPT}{|R|}$. L'algorithme choisit le S_i qui minimise $\frac{w(S_i)}{|S_i \cap R|}$ pour couvrir u_k, et donc le coût de u_k ne dépasse pas $\frac{OPT}{|R|}$. On a la chaîne d'inégalités

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

et c'est tout ce qu'il nous fallait. Facile non?

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.

Solution.

Rappelons que l'algorithme glouton choisit toujours le S_i qui maximise

$|S_i \cap R|$. Comme dans l'analyse du SET-COVER normal, pour chaque $u_j \in U$, on assigne

$$cout(u_j) = \frac{1}{|S_i \cap R|}$$

où S_i est le premier élément qui a couvert u_j et R est l'ensemble des éléments qui restaient à couvrir avant d'ajouter S_i à la solution gloutonne.

Soit $S_{OPT} \subseteq S$ une solution optimale, avec $OPT = |S_{OPT}|$. Considérez un $S_h \subseteq S_{OPT}$ en particulier dans cette solution, avec $S_h = \{u_1, u_2, \ldots, u_k\}$. Soit u_j le premier élément de S_h couvert par l'algorithme glouton. On devra généraliser au p-ème élément couvert, mais on discute du premier pour développer une intuition.

Notez que u_j n'est pas nécessairement couvert par S_h dans l'algorithme glouton. Par contre, à ce moment, puisque u_j est le premier élément de S_h couvert, on avait $|S_h \cap R| = k$, le maximum possible. Il se peut que l'algorithme ait choisi un autre ensemble S_i pour couvrir u_k, mais il faut que $|S_i \cap R| = k$ car l'algorithme maximise cette quantité. Ceci veut dire que

$$cout(u_j) = \frac{1}{|S_i \cap R|} = \frac{1}{k}$$

Ceci n'est pas suffisant, il faut généraliser au p-ème élément de S_h couvert par l'algorithme, pour $p \in \{1, 2, \ldots, k\}$. Disons que u_j est le p-ème élément de S_h couvert par l'algorithme. Au moment de couvrir u_j, il y avait $k - p + 1$ éléments de S_h non-couverts. L'algorithme aurait pu choisir S_h avec $|S_h \cap R| = k - p + 1$, ou bien aurait pu choisir un ensemble S_i avec une plus grande intersection. En d'autres termes, ceci veut dire que si S_i a couvert u_j, on a $|S_i \cap R| \geq k - p + 1$. Donc

$$cout(u_j) = \frac{1}{|S_i \cap R|} \leq \frac{1}{k - p + 1}$$

On déduit que

$$\sum_{u_j \in S_h} cout(u_j) \leq \sum_{p=1}^{k} \frac{1}{k - p + 1} = \sum_{p=1}^{k} \frac{1}{k} = H_k$$

Ceci est vrai pour chaque $S_h \in S_{OPT}$. On a alors

$$\begin{aligned} APP &= \sum_{u_j \in U} cout(u_j) \\ &\leq \sum_{S_h \in S_{OPT}} \sum_{u_j \in S_h} cout(u_j) \\ &\leq \sum_{S_h \in S_{OPT}} H_k \\ &= |S_h| \cdot H_k \\ &= H_k \cdot OPT \end{aligned}$$
