IFT800 - Série d'exercices #1

Manuel Lafond

Exercice 1. Le problème SET-COVER est identique au 3-SET-COVER vu en classe, mais les ensembles S qu'on reçoit peuvent avoir des tailles arbitraires. Soit $k = \max_{S' \in S} |S'|$. Donnez une k-approximation pour ce problème.

Solution.

Ceci est la même chose que ce qui a été fait en classe, mais en remplaçant 3 par k.

Notre k-approximation est la suivante.

fonction *ksetcover*(U, S)
  $S^* = \{\}$
  pour $i = 1..n$ **faire**
    Soit u_i le i-ème élément de U
    si u_i *n'est pas couvert par* S^* **alors**
      Ajouter à S^* n'importe quel ensemble qui contient u_i
  fin
  return S^*

Montrons que c'est une k-approximation. Puisqu'un ensemble $S_i \in S$ couvre au maximum k éléments et qu'il y a n éléments à couvrir, on a

$$OPT \geq \lceil n/k \rceil \geq n/k$$

Dans le pire des cas, notre algo retourne un ensemble par élément de U, et donc $APP \leq n$. On obtient

$$APP \leq n = k \cdot n/k \leq k \cdot OPT$$

Exercice 2. Dans le problème MAX-ACYCLIC-SUBGRAPH, on reçoit un graphe *orienté* et on cherche à conserver un nombre maximum d'arêtes pour que le graphe n'ait plus de cycle.

> MAX-ACYCLIC-SUBGRAPH
> **Entrée :** un graphe orienté $G = (V, E)$
> **Sortie :** un ensemble $E' \subseteq E$ de taille maximum tel que $G' = (V, E')$ n'a aucun cycle.

Donnez une 1/2-approximation pour ce problème.

Indice. Ceci est facile, mais seulement quand on a un indice. Je vous invite à tenter l'exercice sans l'indice, puis à revenir ici une fois que vous aurez échoué.

(à lire après votre échec :) Utilisez $OPT \leq |E|$ comme borne. Ensuite, ordonnez les sommets de G de gauche à droite arbitrairement. Considérez les arêtes qui vont de gauche à droite, ou les arêtes qui vont de droite à gauche. Conservez un de ces deux sous-ensembles.

Solution.

Voici l'algo. L'idée est telle que décrite dans l'indice.

fonction *maxdag*$(G = (V, E))$
  //Rappel : une permutation d'un ensemble est une façon d'ordonner les éléments de l'ensemble
  Soit $P = (v_1, v_2, \ldots, v_n)$ une permutation arbitraire de V
  $E_{fwd} = \{\}$
  $E_{back} = \{\}$
  pour $i = 1..n$ **faire**
    pour $j = i + 1..n$ **faire**
      si $(v_i, v_j) \in E$ **alors**
        $E_{fwd}.append((v_i, v_j))$
      si $(v_j, v_i) \in E$ **alors**
        $E_{back}.append((v_j, v_i))$
    fin
  fin
  si $|E_{fwd}| \geq |E|/2$ **alors**
    return E_{fwd}
  sinon
    return E_{back}

On va démontrer que c'est une 1/2-approximation. L'idée est dans la figure.

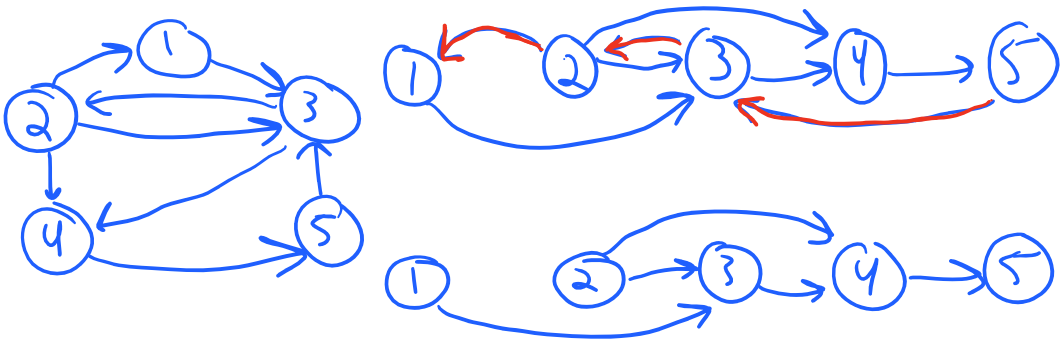

Figure 1: À gauche, un exemple d'instance. À droite en haut, le même graphe mais représenté après avoir ordonné les sommets. Les arêtes bleues sont E_{fwd} et les arêtes rouges sont E_{back}. En dessous, la solution $G = (V, E_{fwd})$.

On sait bien sûr que

$$OPT \leq |E|$$

Soit E' la solution retournée par l'algo. Il est clair que $G = (V, E')$ n'a pas de cycle: si l'algo retourne E_{fwd}, on n'a que des arêtes qui vont vers la droite, rendant un cycle impossible dans $G = (V, E_{fwd})$. L'argument est le même si l'algo retourne E_{back}.

De plus, si l'algo retourne E_{fwd}, on a

$$APP = |E_{fwd}| \geq |E|/2 \geq OPT/2$$

Si l'algo retourne E_{back}, on a $E_{back} = |E| - |E_{fwd}| \geq |E| - |E|/2 \geq |E|/2$. Encore une fois,

$$APP = |E_{back}| \geq |E|/2 \geq OPT/2$$

Exercice 3. Considérez l'algorithme glouton pour VERTEX-COVER. Cet algorithme trouve le sommet de degré maximum, l'ajoute à la solution, puis retire le sommet ainsi que toutes ses arêtes. On répète cette procédure jusqu'à ce qu'il n'y ait plus d'arête. Démontrez que ce n'est *pas* une 2-approximation.
Défi. Montrez qu'il existe des instances dans lesquelles cet algorithm retourne une solution telle que $APP \geq O(\log(n))OPT$.
Défi. Montrez que c'est une $O(\log n)$-approximation.

Solution. Cette question était sans doute une des plus difficiles. Pour montrer que l'algorithme glouton n'est PAS une 2-approximation, on montre un exemple d'instance où $APP > 2 \cdot OPT$. Voir la figure.

On construit G avec 12 sommets de base, qui apparaissent en noir sur la figure. Appelons cet ensemble de 12 sommets X. On ajoute ensuite :
2 autres sommets, chacun ayant 6 voisins distincts dans X (en rouge);
3 autres sommets ayant chacun 4 voisins distincts dans X (en rose);
4 autres sommets ayant chacun 3 voisins distincts dans X (en vert);
6 autres sommets ayant chacun 2 voisins distincts dans X (en bleu);
12 autres sommets ayant chacun 1 voisin distinct dans X (en orange).

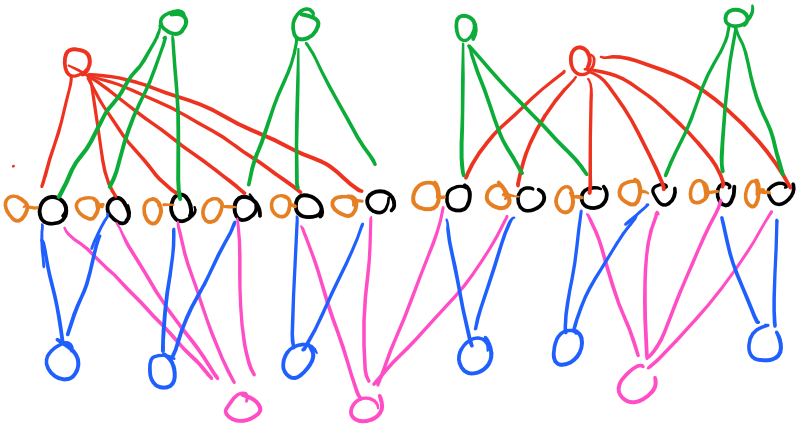

Figure 2: Une instance sur laquelle l'algo glouton donne $APP > 2OPT$.

L'algo ajoutera d'abord à la solution les sommets rouges (degré max). Ensuite, ces sommets rouges seront retirés du graphe. Les sommets de X auront donc un degré de 4, à égalité avec les roses. L'algo pourrait donc décider d'ajouter à la solution tous les sommets roses. Après les avoir retirés, X tombe à degré 3. L'algo pourrait décider d'ajouter les sommets verts. Puis l'algo ajouterait les sommets bleus, puis les oranges.

Au total, l'algo ajoute $APP = 2 + 3 + 4 + 6 + 12 = 27$ sommets, alors que $OPT = 12$. On a $APP = 27/12 \cdot OPT > 2 \cdot OPT$. Cet exemple est suffisant pour montrer que l'algorithme n'est PAS une 2-approximation.

Pour répondre au premier défi 1, il faut généraliser cette construction et créer n sommets de base. Ensuite, pour chaque i, on ajoute i sommets ayant $\lfloor n/i \rfloor$ sommets distincts dans la base. L'analyse se complique un peu, mais finit par donner $APP \geq \Omega(\log n)OPT$. Je vous laisse regarder.

Pour le deuxième défi, on peut imiter l'analyse de SET-COVER, qui viendra plus tard dans la session.

Exercice 4. Montrez que la 2-approximation pour VERTEX COVER des notes de cours est *serrée*.

Solution.

Pour montrer que c'est serré, on montre un exemple d'instance où $APP = 2 \cdot OPT$, où APP est la nombre de sommets de la solution retournée par l'algorithme qui utilise les matchings.

Il suffit d'un graphe G avec deux sommets et une arête. Un matching retournera l'arête et l'approximation va inclure $APP = 2$ sommets, alors que $OPT = 1$. Donc, on a un exemple d'instance où $APP = 2 \cdot OPT$.

Exercice 5. Montrez que la 1/2-approximation pour MAX-SAT des notes de cours est *serrée*.

Solution.

On prend une instance sur variables x_1, x_2 avec deux clauses triviales: $C_1 = x_1$ et $C_2 = \overline{x}_2$.

L'algo va retourner $x_1 = x_2 = True$ et satisfaire $APP = 1$ clause, alors que clairement, $OPT = 2$. On a $APP = 1/2 \cdot OPT$.

Exercice 6. Montrez que la 2-approximation pour COMMIS-VOYAGEUR des notes de cours est *serrée* (celle-ci est moins facile).

Solution.

On crée une instance avec n sommets $v_1, v_2, \ldots, v_n$ telle que $f(v_i v_{i+1}) = 2$ pour tout $i \in \{2, \ldots, n-1\}$, et $f(v_i v_j) = 1$ sinon.

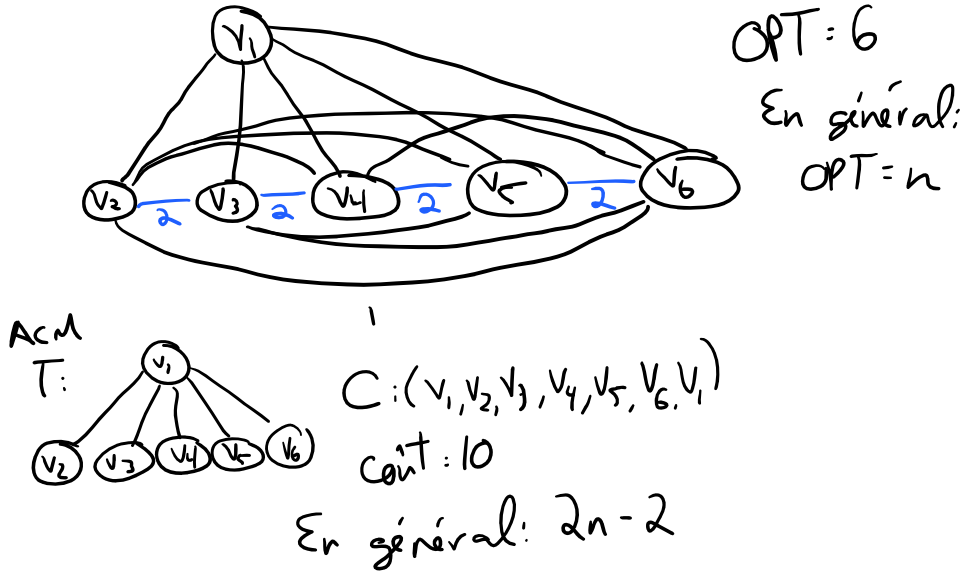

Figure 3: Un exemple de mauvaise instance pour l'algorithme du commis voyageur vu en classe. Les arêtes noires sont de poids 1.

Il n'est pas difficile de voir que nos instances forment des métriques et que $OPT = n$. Il se pourrait que l'algorithme construise un ACM avec v_1 à la racine, et enfants $v_2, v_3, \ldots, v_n$. Avec un parcours pré-ordre, l'algo retourne le cycle $C = (v_1, v_2, \ldots, v_n, v_1)$ de coût $2n - 2$.

Pour connaître le facteur d'approximation c, on a

$$APP = 2n - 2 = c \cdot n = c \cdot OPT$$

En isolant c, on trouve $c = 2 - 2/n$, ce qui tend vers 2 lorsque n tend vers l'infini. On déduit qu'il existe des instances montrant que $APP \geq (2-\epsilon) \cdot OPT$ pour tout $\epsilon > 0$.

Exercice 7. Soit $k \in \mathbb{N}$ une **constante**. On reçoit un graphe $G = (V, E)$ et on nous garantit qu'il existe une couverture par sommet contenant k sommets ou moins (donc un ensemble $X \subseteq V(G)$ de taille k ou moins qui touche chaque arête). Donnez une 1-approximation pour ce problème.

Indice. Combien de sous-ensembles de k sommets y a-t-il?

Solution.

Il suffit de remarquer que k est une constante, et donc qu'un algo $O(n^k)$ est polynomial. Notre algo retourne la plus petite couverture minimum, avec la condition qu'elle ne doit pas être plus grosse que k (sinon on retourne null).

```
fonction kcouverture(G = (V, E))
    bestX = null
    pour chaque sous-ensemble X ⊆ V de taille k ou moins faire
        couvrant = True
        pour chaque uv ∈ E faire
            si u ∉ X et v ∉ X alors
                couvrant = False
        fin
        si couvrant et (bestX = null ou |X| < |bestX|) alors
            bestX = X
    fin
    return bestX
```

Il y a $\sum_{j=1}^{k} \binom{n}{j}$ sous-ensembles de taille k. Sachant que k est une constante, pour n assez grand, on a

$$\sum_{j=1}^{k} \binom{n}{j} \leq k\binom{n}{k} \in O(n^k)$$

L'algorithme prend donc un temps $O(n^k|E|)$ et est en temps polynomial, puisque k est une constante. De plus, $APP = OPT$ car on teste chaque sous-ensemble possible. On a donc une 1-approximation.

Exercice 8. Considérez le problème SET-COVER avec une borne sur le nombre d'occurrences des éléments.

SET-COVER-k-OCC
Entrée : un univers $U = \{u_1, \ldots, u_n\}$ et des ensembles $S = \{S_1, \ldots, S_m\}$ de tailles arbitraires tels que pour chaque u_i, il y a au plus k ensembles qui le contiennent
Sortie : un sous-ensemble $S^* \subseteq S$ de taille minimum qui couvre U.

Nous allons développer une k-approximation pour ce problème.

1. Deux éléments u_i, u_j sont appelés indépendants s'il n'existe pas d'ensemble $S_l \in S$ tel que $u_i \in S_l$ et $u_j \in S_l$.

   Soit $X \subseteq U$ tel que chaque paire d'éléments de X sont indépendants. Montrez que $OPT \geq |X|$.

2. Considérez la stratégie suivante: trouver un ensemble X d'éléments indépendants, puis ajouter à votre solution S^* tous les ensembles de S qui contiennent un élément de X.

   Il faut argumenter que trouver un X approprié peut s'implémenter en temps polynomial, donne une solution faisable et donne une solution avec $|S^*| \leq k \cdot OPT$.

   Indice. Trouvez un X indépendant maximal, c'est-à-dire que peu importe quel élément u_j on tente de lui ajouter, $X \cup \{u_j\}$ n'est plus indépendant. Notez que maximal ne veut pas dire maximum.

Solution.

Pour 1), soit X un ensemble d'éléments indépendants. Puisqu'aucun ensemble de S ne contient deux éléments de X, tout ensemble couvrant doit contenir un élément distinct par élément de X, et donc $OPT \geq |X|$.

Pour 2), voici notre k-approximation.

```
fonction setcover-kapprox(S, U)
    X = {}
    //Étape 1 : construire un ensemble indépendant
    pour u_i ∈ U faire
        estIndep = True
        pour chaque S_j ∈ S qui contient u_i faire
            si S_j contient au moins un élément de X alors
                estIndep = False
        fin
        si estIndep alors
            X.append(u_i)
    fin
    //Étape 2 : retourner tous les S_j avec un élément de X
    S* = {}
    pour u_i ∈ X faire
        pour S_j ∈ S tel que u_i ∈ S_j faire
            S*.append(S_j)
        fin
    fin
    return S*
```

On montre d'abord que S^* couvre U. Supposons que S^* ne couvre pas U. Alors il existe $u_i \in U$ tel que $u_i \notin S_j$ pour tout $S_j \in S^*$. Donc, aucun ensemble de S ne contient à la fois u_i et un élément de X, car S^* contient tous les ensembles contenant un élément de X. Ceci est une contradiction, car l'algorithme aurait ajouté u_i à X à l'étape 1. Donc, on suppose que S^* couvre U.

On montre que c'est une k-approximation. On sait que

$$OPT \geq |X|$$

Puisque chaque $u_i \in X$ est contenu dans au plus k ensembles, notre algo ajoute au plus k ensembles pour chaque $u_i \in X$ à l'étape 2. Donc

$$APP \leq k|X| \leq kOPT$$

Exercice 9. Soit $G = (V, E)$ un graphe. Un *triangle* est un cycle de longueur 3. On cherche à retirer un nombre minimum d'arêtes pour que G n'ait plus de triangle. Dit autrement, trouver $E' \subseteq E$ de taille minimum tel que $G' = (V, E \setminus E')$ ne contient pas de triangle.

Donnez une 3-approximation pour ce problème.

Indice. Soit U l'ensemble de tous les triangles. Pour chaque arête e, soit S_e l'ensemble des triangles desquels e fait partie. Utilisez le résultat de l'exercice précédent.

Solution.

L'idée est de transformer ce problème en SET-COVER. On veut choisir une arête par triangle et les enlever. On veut donc un nombre minimum d'arêtes qui couvrent chaque triangle. Soit U l'ensemble de tous les triangles. Pour chaque arête $uv \in E$, on aura un ensemble S_{uv} qui contient la liste des triangles qui seront éliminés en enlevant l'arête uv. Le problème devient de trouver un minimum de S_{uv} pour couvrir chaque triangle de U.

fonction *triangles*$(G = (V, E))$
  pour chaque $uv \in E$ **faire**
    $S_{uv} = \{\}$ `// Initialisation des` S_{uv}
  fin
  $U = \{\}$
  pour $a, b, c \in V$ **faire**
    si a, b, c *forment un triangle* **alors**
      $U.append(\{a, b, c\})$
      $S_{ab}.append(\{a, b, c\})$
      $S_{ac}.append(\{a, b, c\})$
      $S_{bc}.append(\{a, b, c\})$
  fin
  $S = \{S_{uv} : uv \in E\}$
  $S^* =$ setcover-kapprox(S, U) `// Voir exercice précédent`
  retourner les arêtes correspondantes à S^*

Soit E' la solution retournée par l'algo. On remarque que si on enlève les arêtes de E', on enlève effectivement une arête par triangle. De plus, enlever des arêtes ne peut pas créer de nouveaux triangles, donc E' est une solution valide.

On observe aussi que chaque triangle $\{a, b, c\}$ apparaît dans 3 ensembles de

S, une fois par arête. Donc, on a une instance de SET-COVER dans laquelle chaque élément est contenu dans 3 ensembles. Par le résultat de l'exercice précédent, on a une 3-approximation.

Exercice 10. Considérez le problème MAX-COMMIS-VOYAGEUR, dans lequel on reçoit un graphe complet $G = (V, E)$ et une fonction de poids f sur les arêtes qui est arbitraire, donc qui n'est *pas nécessairement* une métrique. Le but est de trouver un cycle Hamiltonien de poids *maximum*. Donnez une 1/2-approximation pour ce problème.

Suggestion. Il est possible de trouver un matching de poids maximum en temps polynomial.

Solution.

L'idée est de d'abord trouver un matching, puis d'ajouter des arêtes de façon arbitraire à ce matching pour avoir un cycle Hamiltonien.

Pour simplifier, on va supposer ici que n est pair, donc chaque sommet sera présent dans un matching de poids maximum.

fonction *maxCommis*$(G = (V, E), f)$
  $M =$ matching de poids maximum dans G
  Soit $M = \{u_1v_1, u_2v_2, \ldots, u_{n/2}v_{n/2}\}$ (ordre arbitraire)
  return $C = (u_1, v_1, u_2, v_2, \ldots, u_{n/2}, v_{n/2}, u_1)$

La figure ci-bas illustre l'analyse. Soit M un matching de poids maximum. Soit C^* un cycle Hamiltonien de poids maximum (donc $OPT = poids(C^*)$). On remarque que les arêtes de C^* sont formées par l'union de deux matchings M_1 et M_2. Puisque M est maximum, on a

$$OPT = poids(M_1) + poids(M_2) \leq poids(M) + poids(M) = 2poids(M)$$

Le cycle C retourné par l'algo a un poids de $poids(M)$ ou plus, car C contient chaque arête de M. Donc

$$APP \geq poids(M) \geq OPT/2$$

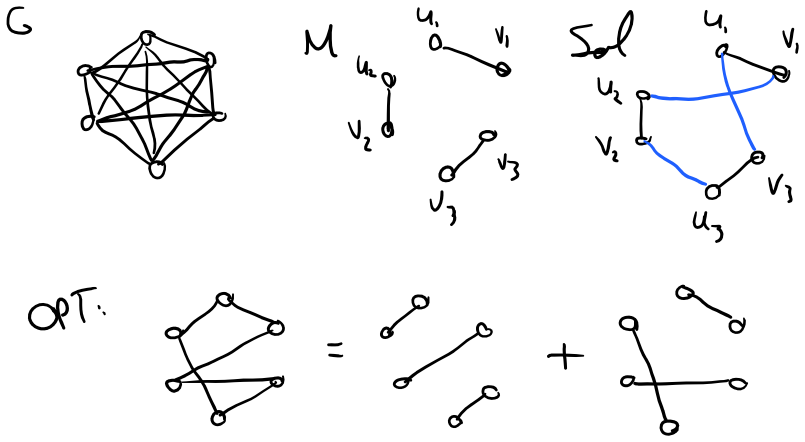

Figure 4: À gauche en haut, un exemple d'instance (poids non-illustrés). À droite en haut, un matching et un cycle qui utilise ce matching. En bas : une solution optimale est une combinaison de deux matchings.
