TD 13 – Approximations-2 (corrigé)

(SubsetSum) **Exercice 1.** *Subset Sum*

Dans un TD précédent, on s'est intéressé au problème de décision SUBSET-SUM consistant à savoir s'il existe un sous-ensemble d'entiers de S dont la somme vaut exactement t. Le problème d'optimisation qui lui est associé prend aussi en entrée un ensemble d'entiers strictement positifs S et un entier t, il consiste à trouver un sous-ensemble de S dont la somme est la plus grande possible sans dépasser t (cette somme qui doit donc approcher le plus possible t sera appelée *somme optimale*).
On suppose que $S = \{x_1, x_2, \dots, x_n\}$ et que les ensembles sont manipulés sous forme de listes triées par ordre croissant. Pour une liste d'entiers L et un entier x, on note $L + x$ la liste d'entiers obtenue en ajoutant x à chaque entier de L. Pour les listes L et L', on note **Fusion**(L, L') la liste contenant l'union des éléments des deux listes.

Premier algorithme

Algorithm 1: Somme(S, t)

début
  $n \leftarrow |S|$;
  $L_0 \leftarrow \{0\}$;
  pour i **de** 1 **à** n **faire**
    $L_i \leftarrow$ **Fusion**$(L_{i-1}, L_{i-1} + x_i)$;
    Supprimer de L_i tout élément supérieur à t;
  retourner *le plus grand élément de* L_n;

1. Quelle est la distance entre la valeur retournée par cet algorithme et la somme optimale ?

   ☞ On a un ensemble $S = \{x_1, \dots, x_n\}$ d'entiers strictement positifs ainsi qu'un entier t. On cherche une somme partielle d'éléments de S maximale et inférieure à t. On appellera somme optimale ce nombre.

   Dans le premier algorithme, L_i représente l'ensemble des sommes partielles des éléments de $\{x_1, \dots, x_i\}$ inférieures à t. L_n représente donc l'ensemble des sommes partielles de S inférieures à t. Par conséquent, $\max(L_n)$ renvoie exactement la somme optimale de S. De plus, cet algorithme a testé toutes les sommes partielles inférieures à t.

   La différence entre le résultat trouvé et la somme optimale est 0.

2. Quelle est la complexité de cet algorithme dans le cas général ? Et si pour un entier $k \geq 1$ fixé, on ne considère que les entrées telles que $t = \mathcal{O}(|S|^k)$, quelle est la complexité de cet algorithme ?

   ☞
   — Dans le pire des cas, on teste toutes les sommes partielles de S soit une complexité en $O(2^n)$: il existe des entrées telles que $|L_i| = 2^i$, prendre par exemple $S = \{1, 2, 2^2, \dots, 2^i, \dots, 2^{n-1}\}$ et t grand ($\geq 2^n$). Or comme la fusion à chaque étape est en $O(|L_{i-1}|)$, la complexité totale est exponentielle en n.
   — Si $t = O(|S|^k) = O(n^k)$ pour k un entier fixé, alors $|L_i| = O(|S|^k)$. Au pas i de la boucle, la fusion et la suppression se font en $O(|L_i|)$. Comme il y a n pas dans la boucle, la complexité totale est $O(|S|^{k+1})$.

Deuxième algorithme Cet algorithme prend en entrée un paramètre ϵ en plus, où ϵ est un réel vérifiant $0 < \epsilon < 1$.

Algorithm 2: Somme-avec-seuillage(S, t, ϵ)

début
  $n \leftarrow |S|$;
  $L_0 \leftarrow \{0\}$;
  pour i **de** 1 **à** n **faire**
    $L_i \leftarrow$ **Fusion**$(L_{i-1}, L_{i-1} + x_i)$;
    $L_i \leftarrow$ **Seuiller**$(L_i, \epsilon/2n)$;
    Supprimer de L_i tout élément supérieur à t;
  retourner *le plus grand élément de* L_n;

L'opération **Seuiller** décrite ci-dessous réduit une liste $L =< y_0, y_1, \ldots, y_m >$ (supposée triée par ordre croissant) avec le seuil δ :

Algorithm 3: Seuiller(L, δ)

```
début
    m ← |L|;
    L' ←< y_0 >;
    dernier ← y_0;
    pour i de 1 à m faire
        si y_i > (1 + δ)dernier alors
            Insérer y_i à la fin de L';
            dernier ← y_i;
    retourner L';
```

3. Évaluer le nombre d'éléments dans L_i à la fin de la boucle. En déduire la complexité totale de l'algorithme. Pour donner la qualité de l'approximation fournie par cet algorithme, borner le ratio valeur retournée/somme optimale.

   ☞ Considérons une liste l = $\{y_0, \ldots, y_m\}$ d'entiers triés par ordre croissant et $\delta > 0$. L'opération de seuillage va consister à dire : soit y' le dernier élément gardé de l (on garde y_0 au départ) si $y_i > y' * (1 + \delta)$ alors on va garder y_i. En gros, on enlève les éléments qui sont trop proches d'autres éléments.

   L'algorithme proposé va fonctionner de la même façon que le précédent sauf qu'il va seuiller à chaque pas de la boucle L_i par rapport à $\epsilon/2n$. On a alors $L_n = \{y_0, \ldots, y_m\}$. On va essayer de majorer m. On sait que :
   $t \geq y_m \geq (1 + \epsilon/2n)^{m-1} * y_0 \geq (1 + \epsilon/2n)^{m-1}$

   D'où : $m \leq \ln t / \ln(1 + \epsilon/2n) + 1 \leq \frac{2n \log(t)}{\epsilon \log(2)} + 1$, car $(m-1)\log(1 + \epsilon/2n) \leq \log(t)$ et donc $(m-1)\epsilon/2n \log(2) \leq \log(t)$ (car on a l'inégalité $x \log(2) \leq log(1 + x)$ quand $0 \leq x \leq 1$).
   m est donc polynomial en $\ln t$, en $1/\epsilon$ et en n, donc polynomial en la taille des données. L'algorithme est en $O(mn)$, donc polynomial en la taille des données. Complexité totale : $O(n \times \left(\frac{2n \log(t)}{\epsilon \log(2)} + 1\right)) = O(\frac{n^2 \log(t)}{\epsilon})$.

   Il faut maintenant vérifier qu'on a bien trouvé une $(1+\epsilon)$Approximation en montrant que : $max\{L_n\} \geq (1+\epsilon) * Opt$(somme optimale)
   On note P_i l'ensemble des sommes partielles de $\{x_1, \ldots, x_i\}$ inférieures à t.
   Invariant : $\forall x \in P_i, \exists y \in L_i,\ x/(1+\delta)^i \leq y \leq x$
   Par récurrence :
   — $i = 0$: OK
   — On suppose que c'est vrai pour $(i-1)$ et on le montre pour i : Soit $x \in P_i$
     $P_i = P_{i-1} \cup (P_{i-1} + x_i)$
     $x = x' + e$ avec $x' \in P_{i-1}$, $e = 0$ *ou* $e = x_i$
     $x' \in P_{i-1}$ donc $\exists y'$, $x'/(1+\delta)^i \leq y' \leq x'$
     Si y' + e est conservé par le seuillage :
     $(x' + e)/(1+\delta)^i \leq x'/(1+\delta)^{i-1} + e \leq y' + e \leq x' + e = x$
     Si y' + e n'est pas conservé par le seuillage :
     $\exists y'' \in L_i,\ y'' \leq y' + x_i \leq y'' * (1 + \delta)$
     $(y' + e)/(1+\delta) \leq y'' \leq y' + e \leq x' + e \leq x$
     $(x'/(1+\delta)^{i-1} + e)/(1+\delta) \leq y''$
     $(x' + e)/(1+\delta)^i \leq y''$
     C'est l'encadrement recherché.
     Grâce à l'invariant, on obtient : $Algo \geq Opt/(1 + \epsilon/2n)^n \geq Opt(1 + \epsilon)$
     On a donc une $(1+\epsilon)$approximation de Subset-sum polynomiale en la taille des données et polynomiale en $1/\epsilon$.

(KCentre2) **Exercice 2.** *K-Centre*

On rappelle quelques définitions :

— Dans un graphe $G = (V, E)$, un *ensemble indépendant* est un sous-ensemble de sommets V' non reliés par des arêtes (si $u \in V'$ et $v \in V'$, alors $(u, v) \notin E$).
— Dans un graphe $G = (V, E)$, un *ensemble dominant* est un sous-ensemble de sommets V' tel que tout sommet de $V \setminus V'$ est adjacent à un sommet de V'. On note $dom(G)$ le cardinal minimal d'un ensemble dominant.

On supposera dans la suite qu'on sait que ces deux problèmes sont NP-Complets.

Soit $G = (V, E)$ un graphe non-orienté, complet, dont les arêtes sont pondérées par une fonction de poids w qui vérifie l'inégalité triangulaire : $w(u, v) \leq w(u, w) + w(w, v)$ pour tout triplet de sommets (u, v, w). Soit aussi un entier $k \geq 1$.
Pour tout $S \subset V$ et tout $v \in V \setminus S$, on définit $connect(v, S)$ comme le poids minimal d'une arête reliant v à un sommet de S : $connect(v, S) = \min_{s \in S}\{w(v, s)\}$. Le problème est de trouver un k-centre, c'est à dire un sous-ensemble S de cardinal k et tel que $center(S) = \max_{v \in V \setminus S}\{connect(v, S)\}$ soit minimal.

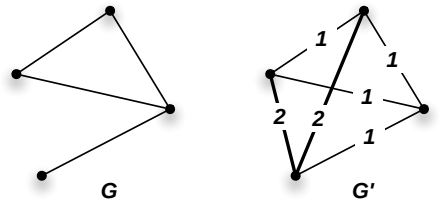
1. À quoi peut bien servir de déterminer un k-centre (donner un exemple d'application) ?

☞ Prenons un ensemble de villes, avec les distances entre ces villes connues. On veut choisir un sous-ensemble de k villes pour implanter des entrepôts, de façon à minimiser la distance maximum entre une ville et l'entrepôt le plus proche.

Autres exemples : arrêts de bus, serveur de pages Web, et bien sûr machines à café...

2. Montrer que trouver un k-centre est NP-difficile.

☞ Nous allons réaliser la réduction à partir d'un ensemble dominant telle que si Ensemble Dominant répond "oui" alors k-centre aura pour solution 1, et sinon (si Ensemble Dominant répond "non") alors on aura 2 pour solution.
Soit une instance du problème Ensemble Dominant $(G = (V,E),k)$, on construit une nouvelle instance pour k-centre de la façon suivante. À partir de G on construit un graphe complet $G' = (V,E')$ avec pour fonction de poids pour $e \in E'$ $w(e) = 1$ si $e \in E$, et $w(e) = 2$ sinon. Puisque toutes les distances dans G' valent 1 ou 2, on a nécessairement que la valeur du k-centre est soit 1 soit 2.
Si on a un certificat D pour Ensemble Dominant, de taille k, alors il nous suffit de placer un "centre" sur chacun des sommets de D pour avoir que chaque $v \in V \backslash D$ est à une distance 1 de D dans G', et la solution pour le k-centre est alors 1.
Inversement, si k-centre retourne 1 comme valeur, alors on a un ensemble $S \subseteq V$ tel que tout sommet $v \in V \backslash S$ soit à une distance 1 de S. Par construction, tous ces sommets doivent avoir une arête $e \in E$ vers un sommet de S, et donc S est un ensemble dominant de taille k.



On va chercher une 2-approximation, i.e. un S de cardinal k tel que $center(S) \leq 2 \cdot OPT$, où $OPT = \min_{S \subset V, |S|=k}\{center(S)\}$.
On ordonne les arêtes de E par poids croissant : $w(e_1) \leq w(e_2) \leq \ldots \leq w(e_m)$, où $m = |E|$. On pose $G_i = (V, E_i)$ où $E_i = \{e_1, e_2, \ldots, e_i\}$ est l'ensemble des i premières arêtes.

3. Montrer que résoudre le problème du k-centre revient à trouver le plus petit indice i tel que G_i a un ensemble dominant de cardinal au plus k.

☞ On veut minimiser $center(S)$.
On veut montrer qu'avoir S un ensemble dominant de G_i est équivalent à $center(S) \leq w(e_i)$.

Soit S un ensemble dominant de G_i
$\iff \forall v \in V \backslash S, \exists s \in S | (v,s) \in E_i$
$\iff \forall v \in V \backslash S, \exists s \in S | w(v,s) \leq w(e_i)$
$\iff \forall v \in V \backslash S, connect(v,S) \leq w(e_i)$
$\iff center(S) = \max_{v \in V \backslash S}\{connect(v,S)\} \leq w(e_i)$

Donc minimiser $center(S)$ avec $|S| \leq k$ revient à trouver le plus petit $w(e_i)$ (c'est à dire l'indice i minimum, car les $w(e_i)$ sont triés par ordre croissant), tel que G_i a un ensemble dominant S de cardinal k.

Une dernière définition : le carré d'un graphe $G = (V,E)$, noté $G^{(2)} = (V, E^{(2)})$, contient les chemins de longueur au plus deux : $(u,v) \in E^{(2)}$ si $(u,v) \in E$ ou s'il existe $w \in V$ tel que $(u,w) \in E$ et $(w,v) \in E$.

4. Étant donné un graphe H, soit I un ensemble indépendant du graphe carré $H^{(2)}$. Montrer que $|I| \leq dom(H)$.

☞ On veut montrer que si I est un ensemble indépendant de $H^{(2)}$ (c'est-à-dire $\forall x,y \in I, (x,y) \notin E_H^{(2)}$), alors $|I| \leq dom(H)$.
Soit D un ensemble dominant de H, alors
- d'une part, tout sommet de I est "dominé par" (c'est-à-dire relié ou égal à) au moins un sommet de D dans H (car D ensemble dominant de H).
- d'autre part, tout sommet de D "domine" (est relié ou égal à) au plus un sommet de I dans H (sinon, soient $x,y \in I$ dominés par un même sommet dans H, alors x et y sont à une distance 1 ou 2 dans H, autrement dit (x,y) est une arête de $H^{(2)}$, d'où une contradiction avec I ensemble indépendant de $H^{(2)}$).

Donc $|I| \leq dom(H)$.

5.

L'algorithme d'approximation du k-centre est le suivant : ☞ **Remarque :** Complexité de l'algorithme (non demandée) : ligne 4 en $O(m \times n^3)$ (calcul de $G_i^{(2)}$ à partir de G_i en $O(n^3)$ avec produit de matrices d'adjacence), ligne 4 en $O(m \times n^2)$ (calcul avec un glouton qui pioche un sommet mis dans M_i, puis retire ce sommet et ses voisins dans $G_i^{(2)}$ et recommence sur le graphe restant, possible en $O(n^2)$, ou $O(m+n)$ avec des listes d'adjacences), ligne 4 en $O(m)$. On a donc un algorithme polynomial.

(a) Montrer que $w(e_j) \leq OPT$.

☞ Montrons que $w(e_j) \leq OPT = \min_{S \subseteq V, |S|=k}\{center(S)\}$.
On sait d'après la question 3 que $OPT = \min_i\{w(e_i) | dom(G_i) \leq k\}$, or $\forall i < j, |M_i| > k$ (ligne 3 de l'algo)) $\Rightarrow dom(G_i) \geq |M_i| > k$ (d'après la question 4).

On peut donc écrire : $OPT = \min_{i \geq j}\{w(e_i) | dom(G_i) \leq k\} \geq w(e_j)$, car les $w(e_i)$ sont triés par ordre croissant.

début

Construire $G_1^{(2)}, G_2^{(2)}, \ldots, G_m^{(2)}$;
Trouver de manière gloutonne un ensemble indépendant inextensible (auquel on ne peut pas rajouter des sommets) M_i dans chaque graphe $G_i^{(2)}$;
Trouver le plus petit indice i tel que $|M_i| \leq k$, soit j cet indice;
retourner M_j;

(b) Montrer que l'algorithme est bien une 2-approximation.

☞ **Remarque :** À ce stade, on n'a pas encore utilisé le fait que les M_i, et donc M_j, sont maximum pour l'inclusion, ni l'inégalité triangulaire sur les $w(e_i)$.

☞ Montrons que $center(M_j) \leq 2 \cdot OPT$.

On a $center(M_j) = \max_{v \in V \setminus M_j} \{connect(v, M_j)\}$, soit $v \in V \setminus M_j, \exists s \in M_j$ tel que (v,s) arête de $G_j^{(2)}$ (en effet, sinon M_j ne serait pas un indépendant maximal pour l'inclusion de $G_j^{(2)}$, car on pourrait ajouter v à M_j).

Par conséquent :
— soit (v,s) arête de G_j, donc $w(v,s) \leq w(e_j)$
— soit $\exists$ sommet z tel que (v,z) arête de G_j et (z,s) arête de G_j
Donc $w(v,s) \leq w(v,z) + w(z,s) \leq 2 \cdot w(e_j)$.

Donc, de toute façon, $connect(v, M_j) \leq 2 \cdot w(e_j)$, et au final $center(M_j) = \max\{connect(v, M_j)\} \leq 2 \cdot w(e_j) \leq 2 \cdot OPT$

6. Montrer que la borne 2 est stricte : donner un exemple de graphe où l'algorithme réalise effectivement une 2-approximation.

☞ Un graphe atteignant la borne 2 est par exemple une roue de $n+1$ sommets : chaque arête incidente au point central de la roue a un poids de 1, et toutes les autres arêtes ont un poids de 2.

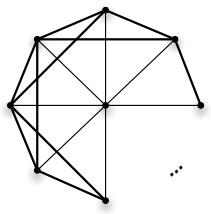


FIGURE 1 – Roue : les arêtes de poids 1 sont représentés en trait fin, et les arêtes de poids 2 en trait fort.

Pour $k = 1$, la solution optimale est le centre de la roue, et $OPT = 1$. L'algorithme va lui calculer l'indice $j = n$. G_n^2 est une clique, et si la solution renvoyée est un des sommets autres que le bord, alors le coût de la solution est 2.

7. Montrer que si $P \neq NP$, il n'existe pas de $(2-\varepsilon)$-approximation au problème du k-centre, pour tout $\varepsilon > 0$.

☞ On va montrer que si un tel algorithme existait, alors il résoudrait le problème en temps polynomial. On va faire une réduction à partir du problème de décision de l'existence d'un ensemble dominant.

Soit $G = (V, E)$, k une instance du problème d'ensemble dominant. On construit alors un graphe complet $G' = (V, E')$, avec pour poids des arêtes :

$$cost(u,v) = \begin{cases} 1, \ if \ (u,v) \in E \\ 2 \ sinon \end{cases}$$

G' satisfait bien l'inégalité triangulaire. La réduction satisfait bien les conditions :
— si $dom(G) \leq k$, alors G' a un k-centre de coût 1
— si $dom(G) > k$, alors le coût optimum d'un k-centre de G' est 2.

Dans ce cas, lorsqu'on utilise l'algorithme de $(2-\epsilon)$-approximation sur le graphe G', il doit renvoyer une solution de coût 1, puisqu'il ne peut pas utiliser une arête de coût 2. On a donc avec cet algorithme un moyen de déterminer en temps polynomial s'il existe ou non un ensemble dominant. Donc si $P \neq NP$ ce n'est pas possible.