TD 13 – Approximations-2

(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 $S = \{x_1, x_2, \ldots, x_n\}$ 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 ← |S|;
    L_0 ← {0};
    pour i de 1 à n faire
        L_i ← 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 ?
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 ?

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 ← |S|;
    L_0 ← {0};
    pour i de 1 à n faire
        L_i ← Fusion(L_{i−1}, L_{i−1} + x_i);
        L_i ← Seuiller(L_i, ε/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.

(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) ?

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

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.

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

5. L'algorithme d'approximation du k-centre est le suivant :

---

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;

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

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

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

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