TD 12 – Approximations

(Nuages) **Exercice 1.** *Clustering*

On considère n points dans un espace métrique (les distances entre les points satisfont l'inégalité triangulaire). On veut partitionner les points en k groupes de manière à minimiser le plus grand diamètre d'un groupe. Le diamètre d'un groupe est la distance maximale entre deux points de ce groupe. Noter que n et k sont fixés dans l'énoncé du problème.

1. On suppose que le diamètre optimal d est connu. Trouver une 2-approximation pour le problème.
2. On considère l'algorithme qui, k fois, choisit comme "centre" le point à distance maximale de tous les centres déjà choisis, puis alloue chaque point au centre le plus proche. En faisant le lien avec la question précédente, montrer que cet algorithme est une 2-approximation pour le problème.

(CouvertureSommets) **Exercice 2.** *Vertex Cover*

Nous considérons ici le problème de la couverture par sommets : étant donné un graphe non-orienté $G = (V, E)$ et une fonction de poids sur les sommets $w : V \to \mathbb{Q}+$, trouver $S \subseteq V$ de poids minimum qui couvre les arêtes (c'est à dire $\forall\, e \in E, \exists\, s \in S : s \in e$). En cours, nous avons vu une 2-approximation dans le cas où w est constante. Nous allons étudier ici une 2-approximation dans le cas général. Pour ce faire nous nous intéressons à un type de fonction de poids particulier : les fonctions de poids par degré. w est une fonction de poids par degré si $\exists c > 0, \forall v \in V : w(v) = c \cdot deg(v)$.

1. Soit $\{G = (V, E), w\}$ une instance du problème telle que w est une fonction de poids par degrés. Montrer que $w(V) \leq 2 \cdot OPT$ où $w(V) = \sum_{v \in V} w(v)$.

   On sait donc traiter les instances avec une fonction de poids par degrés. La méthode du mille-feuille consiste alors à décomposer une instance quelconque en une famille d'instances avec fonction de poids par degrés.
2. À chaque étape de la décomposition, on cherche la plus grande fonction de poids par degrés p inférieure à w. Expliquer comment calculer p.

L'algorithme du mille-feuille est le suivant :

Données:
$G = (V, E)$;
une fonction de poids quelconque w;
début
$t \leftarrow 0$;
$G_0 \leftarrow G$;
$w_0 \leftarrow w$;
tant que $G_t = (V_t, E_t)$ *contient une arête* **faire**
  $D_t \leftarrow \{u \in V_t : deg_t(u) = 0\}$;
  $p_t \leftarrow$ plus grande fonction de poids par degrés inférieure à w_t dans G_t;
  $S_t \leftarrow \{u \in V_t : p_t(u) = w_t(u)\}$;
  $G_{t+1} \leftarrow G_t \setminus (D_t \cup S_t)$;
  $w_{t+1} \leftarrow w_t - p_t$;
  $t \leftarrow t + 1$;
retourner $C = \bigcup_{k=0}^{t-1} S_k$

3. Montrer que l'algorithme termine en temps polynomial.
4. Montrer que l'ensemble C de sommets retournés par l'algorithme est bien une couverture.

5. Pour tout $v \in C$, exprimer $w(v)$ en fonction des poids par degrés p_k. Qu'en est-il pour $v \notin C$? Remarque : on pourra poser $p_k(u) = 0$ pour tout sommet $u \notin G_k$.

À partir de maintenant, on notera C^* une couverture par sommets optimale pour l'instance de départ $\{G = (V, E), w\}$.

6. Pour toute étape i de l'algorithme, comparer $p_i(C \cap G_i)$ et $p_i(C^* \cap G_i)$.
Indication : on remarquera que $C \cap G_i$ et $C^* \cap G_i$ sont deux couvertures par sommets de G_i.

7. Montrer que l'algorithme est une 2-approximation du problème de la couverture par sommet minimum avec des poids arbitraires. Trouver un exemple où l'algorithme renvoie effectivement une solution de poids $2.OPT$.

(SubsetSum) **Exercice 3.** *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
  $n \leftarrow |S|$;
  $L_0 \leftarrow \{0\}$;
  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 ?

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
  $n \leftarrow |S|$;
  $L_0 \leftarrow \{0\}$;
  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.