TD 04 – Algorithmes gloutons (corrigé)

(Matroide) **Exercice 1.** *Matroides*

Définition. Soit S un ensemble fini et $\mathcal{I}$ une famille de parties de S. Alors $(S, \mathcal{I})$ est un *matroïde* si
— hérédité : pour tout $X \in \mathcal{I}$, pour tout $Y \subset X$, $Y \in \mathcal{I}$;
— échange : $\forall X, Y \in \mathcal{I}$ tels que $|X| < |Y|$, $\exists x \in Y \setminus X$ tel que $X \cup \{x\} \in \mathcal{I}$.
Les éléments de $\mathcal{I}$ sont appelés les *indépendants* du matroïde.

1. Montrer que si $(S, \mathcal{I})$ est un matroïde et T une partie de S, alors si X et Y sont deux indépendants de S inclus dans T et qu'ils sont maximaux pour l'inclusion dans T, alors $|X| = |Y|$. ☞ Supposons que $|X| < |Y|$. Alors par l'échange il existe $x \in Y \setminus X$ tel que $X \cup \{x\} \in \mathcal{I}$. Comme $X \cup Y \subset T$, alors $X \cup \{x\} \subset T$, d'où une contradiction.

Soit S un ensemble fini et $\mathcal{I}$ un ensemble de parties de S. Une *fonction de coût* pour S est une fonction $c : S \to \mathbb{R}_+$. Elle est naturellement étendue à $\mathcal{I}$ en posant $c(X) = \sum_{x \in X} c(x)$. On considère l'algorithme suivant :

Algorithme : `Glouton`$(S, \mathcal{I}, c)$

```
Ordonner les éléments de S = {s_1, ..., s_n} par coût décroissant
X ← ∅
pour i de 1 à n faire
    si X ∪ {s_i} ∈ I alors
        X ← X ∪ {s_i}
```

2. Montrer que si `Glouton` trouve un ensemble $X \in \mathcal{I}$ de coût maximal quelque soit la fonction de coût c, alors $(S, \mathcal{I})$ est un matroïde. ☞ On montre que si `Glouton` fonctionne, alors $(S, \mathcal{I})$ vérifie l'hérédité, puis on montre que si $(S, \mathcal{I})$ ne vérifie pas l'échange, alors `Glouton` ne fonctionne pas.
Supposons que `Glouton` fonctionne. Soit donc $X \in \mathcal{I}$ et Y un sous-ensemble de X. Considérons la fonction de coût c telle que $c(y) = 2$ pour tout $y \in Y$, $c(x) = 1$ pour tout $x \in X \setminus Y$, et $c(s) = 0$ pour les autres éléments. Clairement, l'ensemble optimal est X. Si `Glouton` renvoie X, il doit avoir ajouté un à un ses éléments, en commençant par ceux de Y (puisque leur coût est supérieur). Ainsi, à un moment donné de l'algorithme, la variable X contiendra Y et le test $X \in \mathcal{I}$? sera positif. Autrement dit, $Y \in \mathcal{I}$.

   Supposons que $(S, \mathcal{I})$ ne vérifie pas l'échange, et soit X, Y les parties qui ne fonctionnent pas. Alors on définit $c(s) = |X| + 2$ si $s \in X$, $c(s) = |X| + 1$ si $s \in Y \setminus X$ et $c(s) = 0$ partout ailleurs. La partie de coût maximal est Y qui vaut $|Y|(|X| + 1)$. Or l'algorithme va retourner X, dont le coût est $|X|(|X| + 2) = (|X| + 1)^2 - 1 \leq c(Y) - 1$.

3. Soit $(S, \mathcal{I})$ un matroïde. On considère une fonction de coût c ainsi qu'un ensemble de coût maximal $X_{\text{opt}} \in \mathcal{I}$. On suppose que `Glouton` renvoie X avec $c(X) < c(X_{\text{opt}})$.
   (a) Montrer qu'on peut supposer que $|X| = |X_{\text{opt}}|$. ☞ Par construction, X est maximal au sens de l'inclusion (tant qu'on peut rajouter des gens, on le fait). Maintenant, on peut supposer que $X_{\textbf{opt}}$ l'est également : en effet, si on peut ajouter des éléments, ça ne peut pas faire diminuer le coût. En utilisant la question 1, on conclut.
   (b) On note $X = \{x_1, \ldots, x_p\}$ et $X_{\text{opt}} = \{y_1, \ldots, y_p\}$, rangés par coût décroissant. Montrer que $c(x_1) \geq c(y_1)$. ☞ Par construction. Puisque $X \in \mathcal{I}$, l'hérédité montre que $x_1 \in \mathcal{I}$. Et x_1 est un élément de coût maximal.
   (c) Soit i le plus petit indice tel que $c(x_i) < c(y_i)$, et $Y = \{s \in S : c(s) \geq c(y_i)\}$. Montrer que $\{x_1, \ldots, x_{i-1}\}$ est un indépendant maximal pour l'inclusion dans Y. ☞ Supposons qu'il existe $s \in Y \setminus \{x_1, \ldots, x_{i-1}\}$ tel que $\{x_1, \ldots, x_{i-1}, s\} \in \mathcal{I}$. Alors $c(s) \geq c(y_i) > c(x_i)$ mais ça contredit le fonctionnement de `Glouton`.
   (d) Conclure. ☞ On a alors $c(x_i) \geq c(y_i)$ pour tout i, donc $c(X) \geq c(X_{\textbf{opt}})$. C'est absurde !

(Kruskal) **Exercice 2.** *Algorithme de Kruskal*

Soit $G = (V, E)$ un graphe non orienté. On note $\mathcal{F}$ l'ensemble des forêts de G, c'est-à-dire $\mathcal{F} = \{F \subseteq E$: le graphe (V, F) est sans cycle$\}$.

1. Montrer que $(E, \mathcal{F})$ est un matroïde. ☞ L'hérédité est évidente.

   Soit F_1 et F_2 deux forêts de G telles que $|F_1| < |F_2|$. Supposons que G a n sommets. Alors F_1 a $n - |F_1|$ arbres, et F_2 en a $n - |F_2|$. Ainsi, F_2 a moins d'arbres que F_1, donc il existe un arbre T de F_2 qui n'est pas inclus dans un arbre de F_1 (T possède deux sommets u et v qui n'appartiennent pas à un même arbre de F_1). En considérant les sommets situés sur le chemin entre u et v dans T, on peut supposer u et v voisins dans T. Alors clairement $F_1 \cup \{(u, v)\}$ est sans cycle, ce qui prouve l'hérédité.

On suppose qu'on dispose d'une fonction de poids $w : E \to \mathbb{R}_+$ sur les arêtes du graphe.

2. Déduire de la question précédente un algorithme glouton pour calculer un arbre couvrant de poids minimal dans un graphe, c'est-à-dire un ensemble $T \subseteq E$ d'arêtes de poids minimal tel que le graphe (V, T) est un arbre.

(Couverture) **Exercice 3.** *Couverture par intervalles*

Étant donné un ensemble $\{x_1, \ldots, x_n\}$ de n points sur une droite, décrire un algorithme qui détermine le plus petit ensemble d'intervalles fermés de longueur 1 qui contient tous les points donnés. Prouver la correction de votre algorithme et donner sa complexité.

Algorithm 1: Couverture par intervalles

```
début
    Trier dans l'ordre croissant les x_i;
    X ← {x_1, ..., x_n};
    I ← ∅;
☞   tant que X ≠ ∅ faire
        x_k ← min(X);
        I ← I ∪ {[x_k, x_k + 1]};
        X ← X \ [x_k, x_k + 1];
    retourner I;
```

La complexité est $n \log n$ (tri) + n (parcours de X), c'est-à-dire $O(n \log n)$

Théorème 1. *Cet algorithme est optimal.*

Méthode 1 : avec des matroïdes. *Cette méthode ne fontionne pas. Il est impossible d'exhiber un matroïde (on ne peut pas prouver la propriété d'échange)*

Méthode 2 : par récurrence sur $|X|$. *On ajoute un à un les élements. On montre que le premier est bien contenu dans l'ensemble fourni par l'algorithme.*

- **Initialisation :** *Si on a qu'un seul point, l'algorithme renvoie un seul intervalle : c'est une solution optimale.*
- *Soit un recouvrement optimal* $I_{opt} = \{[a_1, a_1 + 1], \ldots, [a_p, a_p + 1]\}$ *avec* $a_1 < \cdots < a_p$ *et le recouvrement donné par notre algorithme* $I_{glou} = \{[g_1, g_1 + 1], \ldots, [g_m, g_m + 1]\}$ *avec* $g_1 < \cdots < g_m$. *Montrons que* $m = p$.
- *On pose* $I = \left(I_{opt} \backslash \{[a_1, a_1 + 1]\}\right) \cup \{[g_1, g_1 + 1]\}$.
  Puisque $g_1 = x_1$, *I est un recouvrement de X car* $a_1 \leq x_1 \Rightarrow a_1 + 1 \leq x_1 + 1 = g_1 + 1$ *donc* $X \cap [a_1, a_1 + 1] \subseteq X \cap [g_1, g_1 + 1]$ *; autrement-dit* $I_{opt} \backslash \{[a_1, a_1 + 1]\}$ *est un recouvrement de* $X \backslash \{[g_1, g_1 + 1]\}$.
- *Par hypothèse de récurrence, sur* $X \backslash [g_1, g_1 + 1]$ *une solution optimale est donnée par glouton :* $\{[g_2, g_2 + 1], \ldots, [g_m, g_m + 1]\}$ *car* $|X| \backslash [g_1, g_1 + 1] \leq n - 1$
- *Comme* $I_{opt} \backslash \{[a_1, a_1 + 1]\}$ *est un recouvrement de* $X \backslash [g_1, g_1 + 1]$ *avec* $p - 1$ *intervalles, on a :*

$$\left.\begin{array}{r} p - 1 \geq m - 1 \Rightarrow p \geq m \\ I_{opt} \text{ optimal sur } X \\ I_{glou} \text{ recouvrement de } X \end{array}\right\} \Rightarrow m \geq p \Bigg\} \Rightarrow m = p$$

Puisque p est le nombre d'intervalles que renvoie un algorithme optimal, et que $m = p$, *alors, notre glouton est optimal.*

(Memoire) **Exercice 4.** *Utilisation de la memoire*

On souhaite enregistrer sur une mémoire de taille L un groupe de fichiers $P = (P_1, \ldots, P_n)$. Chaque fichier P_i nécessite une place a_i. Supposons que $\sum a_i > L$: on ne peut pas enregistrer tous les fichiers. Il s'agit donc de choisir le sous ensemble Q des fichiers à enregistrer.
On pourrait souhaiter le sous-ensemble qui contient le plus grand nombre de fichiers. Un algorithme glouton pour ce problème pourrait par exemple ranger les fichiers par ordre croissant des a_i.
Supposons que les P_i soient ordonnés par taille ($a_1 \leq \ldots \leq a_n$).

1. Écrivez un algorithme (en pseudo-code) pour la stratégie présentée ci-dessus. Cet algorithme doit renvoyer un tableau booléen S tel que $S[i] = 1$ si P_i est dans Q et $S[i] = 0$ sinon. Quelle est sa complexité en nombre de comparaisons et en nombre d'opérations arithmétiques ?

☞ On suppose que les P_i sont ordonnés par taille croissante ($a_1 \leq \cdots \leq a_n$).

Algorithm 2: Plus grand nombre de fichiers

```
début
    Données: a_1, ..., a_n, L, S
    ;
    S ← [];
    taille ← 0;
    pour i de 1 à n faire
        taille_tmp ← taille + a_i;
        si taille_tmp ≤ L alors
            taille ← taille_tmp;
            S[i] ← 1;
        sinon
            S[i] ← 0;
    retourner S;
```

Il nous faut comparer à chaque fois la nouvelle taille ($taille + a_i$) avec L pour vérifier si on peut ajouter le fichier, il faut donc n comparaisons. Il nous faut au plus n additions (calcul de $taille_{tmp}$).

2. Montrer que cette stratégie donne toujours un sous-ensemble Q maximal tel que $\sum_{P_i \in Q} a_i \leq L$.

☞ Soit un sous ensemble Q quelconque de cardinal maximal. Comparons le à la solution gloutonne G. Soit a_j le plus petit élément présent dans G et non dans Q, et a_k un élément dans Q n'étant pas dans G (comme Q est de cardinalité maximal, on est sur de trouver un tel a_k pour chaque a_j à considérer). L'ensemble Q' obtenu en échangeant dans Q a_k par a_j est toujours un ensemble valide (puisque la taille mémoire est inférieure à celle de Q), et de même cardinalité que Q. Par construction, on peut ainsi construire un ensemble G' à partir de Q, optimal. G' ne peut enfin pas avoir plus d'élément que G, par construction de G, donc la solution glouton est optimale.

3. Soit Q le sous-ensemble obtenu. À quel point le quotient d'utilisation $(\sum_{P_i \in Q} a_i)/L$ peut-il être petit ?

☞ Si les a_i sont quelconques, ils peuvent tous être plus grand que L. Le quotient est alors 0. Dans le cas plus intéressant où $a_i \leq L$, le quotient est au pire $\frac{a_1}{L}$ (exemple avec $a_1 = 1$ et $a_2 = L$.

Supposons maintenant que l'on souhaite enregistrer le sous-ensemble Q de P qui maximise ce quotient d'utilisation, c'est-à-dire celui qui remplit le plus de disque. Une approche *gloutonne* consisterait à considérer les fichiers dans l'ordre décroissant des a_i et, s'il reste assez d'espace pour P_i, on l'ajoute à Q.

4. On suppose toujours les P_i ordonnés par taille croissante. Écrivez un algorithme pour cette nouvelle stratégie.

☞ C'est le même que précédemment sauf qu'on part de la fin de la liste

Algorithm 3: Plus grand nombre de fichiers

```
début
    Données: a_1, ..., a_n, L, S
    ;
    S ← [];
    taille ← 0;
    pour i de n à 1 faire
        taille_tmp ← taille + a_i;
        si taille_tmp ≤ L alors
            taille ← taille_tmp;
            S[i] ← 1;
        sinon
            S[i] ← 0;
    retourner S;
```

5. Montrer que cette nouvelle stratégie ne donne pas nécessairement un sous-ensemble qui maximise le quotient d'utilisation. À quel point ce quotient peut-il être petit ? Prouvez-le.

☞ Contre-exemple : $a_1 = \frac{L}{2} + 1$, et $a_2 = a_3 = \frac{L}{2}$. L'algorithme glouton retournera une solution ayant comme quotient $1/2$, alors que le quotient de la solution optimale est 1.

Le minimum du coefficient d'utilisation est $1/2$ si $a_i \leq L$ (0 sinon). En effet, soit P le poids de la liste de l'algorithme glouton. Par définition, comme la somme des poids est plus grande que L, P est strictement plus grand que $L/2$ (soit $a_n > L/2$, soit la somme des éléments de plus grand poids dépasse $L/2$). Donc le quotient est plus grand que $\frac{\frac{L}{2}+1}{L} = \frac{1}{2} + \frac{1}{L} \simeq \frac{1}{2}$. Ce quotient est atteint pour l'exemple précédent.