Correction TD5: Algorithmes gloutons & Matroïdes

Algorithmique

17 décembre 2019

Exercice 1 Question 1 : Exemples d'application :

- En biologie : classification des plantes et des animaux .
- Évaluation du risque sismique : regroupement des épicentres sismiques observés pour identifier les zones dangereuses ;
- Clustering binaire : dans le domaine médical par exemple, ça permet de faire des analyse sur la présence d'une certaine maladie.

Question 2 : Une idée possible est de trier toutes les distances entre pairs (x,y) par ordre décroissant, puis essayer d'éviter les grandes distances, une par une.

Question 3 : Un algorithme possible

Algorithme : `ClusteringGlouton` $(E = \{x_1, x_2, \dots x_n\}, d$

Données : $E = \{x_1, x_2, \dots x_n\}, d$
Résultat : (A, B) : 2-Partition
$A \leftarrow \emptyset$;
$B \leftarrow \emptyset$;
$F \leftarrow Trier\{(x_i, x_j) \mid 1 \leq i < j \leq n\}$ par ordre décroissant sur la distance $d(x_i, x_j)$;
$i \leftarrow 1$;
tant que $|A \cup B| \neq n$ **faire**
    $(x, y) = F[i]$;
    $i \leftarrow i + 1$;
    si $x \in A$ **alors**
        si $y \notin A$ **alors**
            $B \leftarrow B \cup \{y\}$
    sinon
        si $x \in B$ **alors**
            si $y \notin B$ **alors**
                $A \leftarrow A \cup \{y\}$
        sinon
            si $y \in A$ **alors**
                si $x \notin A$ **alors**
                    $B \leftarrow B \cup \{x\}$
            sinon
                si $y \in B$ **alors**
                    si $x \notin B$ **alors**
                        $A \leftarrow A \cup \{x\}$
                sinon
                    % On va séparer x et y. Il y a deux choix possibles : ajouter x à A et y à B ou ajouter x à B et y à A. On évalue d'abord le cout de chaque choix et on évite le pire. ;
                    $coutxA \leftarrow max(d(a, b) \mid (a, b) \in A \cup \{x\} \times A \cup \{x\})$;
                    $coutxB \leftarrow max(d(a, b) \mid (a, b) \in B \cup \{x\} \times B \cup \{x\})$;
                    $coutyA \leftarrow max(d(a, b) \mid (a, b) \in A \cup \{y\} \times A \cup \{y\})$;
                    $coutyB \leftarrow max(d(a, b) \mid (a, b) \in B \cup \{y\} \times B \cup \{y\})$;
                    % on calcule le cout quand on ajoute x à A et y à B $cout_1 \leftarrow max\{coutxA, coutyB\}$
                    % on calcule le cout quand on ajoute x à B et y à A $cout_2 \leftarrow max\{coutxB, coutyA\}$
                    si $cout_1 > cout_2$ **alors**
                        $A \leftarrow A \cup \{y\}$;
                        $B \leftarrow B \cup \{x\}$
                    sinon
                        $A \leftarrow A \cup \{x\}$;
                        $B \leftarrow B \cup \{y\}$
retourner *(A,B)* ;

Complexité : Avec n éléments, il y a $m = \frac{n(n-1)}{2}$ paires possibles. Donc la complexité de l'algorithme proposé est $O(mlog(m) + mn) = O(n^3)$ (car $m \in \Theta(n^2)$). Le premier terme de la complexité représente la complexité du tri effectué au début de l'algorithme. Pour le reste, puisqu'on traite une paire à chaque itération, alors le pire des cas correspond à m itérations. Pour chaque itération, le pire des cas correspond au dernier tratement ou on doit calculer $coutxA, coutxB, coutyA, coutyB$. Ce calcule demande $O(n)$ car A et B contiennent au plus n éléments.

Exercice 2 Question 1

Par exemple : Organisation des séances de TDs à l'INSA pour une seule salle. La salle est la ressource. Une séance de TD est une tâche (qui dure une unité de temps 1h15). Chaque séance peut avoir une date d'échéance (par exemple un tel TD doit être effectué avant 12h15). On cherche à maximiser le nombre de séances effectuées dans une salle tout en respectant les échéances.

Autre exemple très similaire : la réservation de salle à la bibliothèque de l'INSA. Une salle est une ressource. Une demande de réservation est une tache qui est associé à une date d'échéance. Pour une salle, on veut maximiser le nombre de demandes satisfaites.

Autre exemple est le cas d'ordonnancement de tâches unitaires pour un processeur.

Question 2

Évident : Puisque A est un sous ensemble réalisable, alors si on enlève une ou plusieurs taches, il est toujours possible d'ordonner le reste des taches tout en respectant leurs dates d'échéances.

Question 3

$\leftarrow$: Évident

$\rightarrow$: Supposant que A est réalisable. Il existe une séquence valide L de A. Soit la transformation suivante : s'il existe une paire (a_i, a_j) tel que $d(a_i) > d(a_j)$ et a_i apparaît avant a_j dans la séquence L, alors on échange les positions de a_i et a_j. Cette transformation ne viole pas les contraintes d'échéances car : (1)a_j est exécutée en avance par rapport à sa position dans L ; (2) a_i a pris la position de a_j sachant que $d(a_i) > d(a_j)$ donc la date d'échéance de a_i est respectée ; et (3) toutes les autres taches n'ont pas changé de position par rapport à L. Donc en exécutant cette transformation sur n'importe quelle liste valide on trouve toujours une liste valide. Appliquons maintenant cette transformation plusieurs fois consécutives. On va surement s'arrêter quand la liste devient triée par ordre décroissant sur les dates. Cette séquence correspond à S. Donc S est valide.

Question 4

$\rightarrow$ A est réalisable. On montre que $\forall t \in [0, n], N_t(A) \leq t$.

Supposant par l'absurde que $\exists t \in [0, n]$, tel que $N_t(A) > t$. On note A_t l'ensemble des taches de A ayant une date d'échéance $\leq t$. On remarque que toute séquence de A_t est de longueur $N_t(A) > t$. Donc pour n'importe quelle séquence de A_t, la dernière tache est toujours en retard. Donc A est irréalisable (d'où la contradiction).

$\leftarrow$ On a $\forall t \in [0, n], N_t(A) \leq t$. Dans ce cas, la séquence qui ordonne toutes les taches par leurs dates d'échéance est valide et donc A est réalisable.

Question 5

Soit A et B deux sous ensembles réalisables tel que $|B| > |A|$.

— On montre qu'il existe $k < n$ tel que $N_k(B) \leq N_k(A)$ et $\forall t > k, N_t(B) > N_t(A)$.

On a $N_0(A) = N_0(B) = 0$, $N_n(A) = |A|$, $N_n(B) = |B|$. Puisque $|B| > |A|$, alors $N_n(A) < N_n(B)$. Donc nécessairement il existe un indice $0 < z \leq n$ tel que $\forall t \geq z$, $N_t(A) < N_t(B)$. Si on prend l'élément le plus petit qui vérifie cette propriété, et on le note par k^*. Alors, pour $k = k^* - 1$, on a $k < n$, $N_k(B) \leq N_k(A)$ et $\forall t > k, N_t(B) > N_t(A)$.

— On sait qu'il existe $k < n$, tel que $N_k(B) \leq N_k(A)$ et $\forall t > k, N_t(B) > N_t(A)$. Donc nécessairement, il existe une tache t^* dans B avec une échéance égale à $k + 1$. Considérons maintenant l'ensemble $A^* = A \cup \{t^*\}$. Pour $t \leq k$, $N_t(A^*) = N_t(A) \leq t$, et pour $t > k$, $N_t(A^*) = N_t(A) + 1 \leq N_B(t) \leq t$. Et donc en utilisant la question 4, on a A^* réalisable.

Question 6

On montre que (E, I) avec E l'ensemble des taches et I l'ensemble des tâches réalisables est un matroïde.

— E est fini

— I est héréditaire : Soit $A \in I$. Tout sous ensemble de A appartient à I (question 2)

— Propriété d'échange : Soit $A \in I, B \in I$ tel que $|A| < |B|$. D'après la question précédente, $\exists x \in B$ tel que $A \cup \{x\} \in I$

Maintenant qu'on connait que (E, I) est un matroïde, on définit un cout pour les éléments de E pour définir un matroïde pondéré. Attention, le cout n'est pas arbitraire ! il doit refléter la vraie fonction objective de notre problème initial. On cherche un ensemble de taches réalisable de cardinalité maximale. Donc le cout d'un ensemble de taches est simplement le nombre de taches qu'il contient. Et puisque le cout d'un ensemble doit être obtenu en faisant la somme des cout de chaque élément dans l'ensemble, alors on doit définir le coût d'une tache $a_i \in E$ par simplement la constante 1 (et donc par extension le coût d'un ensemble réalisable $A \in I$ est le nombre de taches dans A).

À ce stade, on peut appliquer directement le théorème des matroïdes pour obtenir l'algorithme glouton suivant :

Algorithme : `Glouton` $(E = \{a_1, \dots a_n\}, d(a_1), d(a_1), \dots, d(a_n)$

Données : $d_(a_1), d_(a_1), \dots, d_(a_n$
Résultat : Ensemble réalisable de tâches de cardinalité maximale
début
  $F \leftarrow \{\}$;
  $L \leftarrow Trier(E)$ par poids décroissant ;
  Remarque : puisque tous les poids sont égaux (valeur=1), alors n'importe quelle permutation de E est acceptable ;
  pour $i \in [1..n]$ **faire**
    si $F \cup \{L[i]\} \in I$ **alors**
      $F \leftarrow F \cup \{L[i]\}$;
  retourner F ;

Pour que l'algorithme fonctionne, il faut bien définir le test d'appartenance $F \cup \{L[i]\} \in I$. Pour cela, on peut utiliser la question 4 pour observer qu'il suffit de trier $F \cup \{L[i]\}$ par ordre croissant pour obtenir une liste F'. Puis, tester si $F'[k] \geq k$. Cette approche peut être améliorée si F est une liste déjà triée. Car si F est triée, alors pour construire F', il suffit d'insérer $L[i]$ dans F en faisant un parcours de F à l'envers. On peut avoir une troisième amélioration : remarquons que toutes les tâches ont le même coût. Alors la liste L peut être n'importe quelle permutation de E. Si on choisit une permutation de E qui trie les taches par ordre croissant sur les dates d'échéance, alors on est sûr que $L[i]$ est plus grand que tous les éléments de F. Donc si on représente F par une liste, l'ajout de $L[i]$ à F s'effectue automatiquement à la fin de F. Dans ce cas, le test d'appartenance est équivalent à $L[i] \geq |F + 1|$?.

Algorithme : `GloutonAmélioré` $(E = \{a_1, \dots a_n\}, d(a_1), d(a_1), \dots, d(a_n)$
Données : $d_(a_1), d_(a_1), \dots, d_(a_n$
Résultat : Ensemble réalisable de tâches de cardinalité maximale
début
  $F \leftarrow \emptyset$;
  $L \leftarrow Trier(E)$ par dates d'échéance croissantes ;
  pour $i \in [1..n]$ **faire**
    si $L[i] \geq |F + 1|$ **alors**
      $F \leftarrow F \cup \{L[i]\}$;
  retourner F ;

La complexité de cet algorithme est simplement la complexité du tri $\Theta(nlog(n))$.