

Cours *Conception et analyse d'algorithmes*

TD no 1 – Algorithmes glouton et matroïdes

15 septembre 2010

1. Des limites de la gloutonnerie

Cet exercice a pour but de montrer que de nombreux problèmes ne sont pas résolubles par un algorithme glouton. Considérons par exemple celui dit du sac à dos. On doit emporter en voyage un certain nombre d'objets parmi n , l'objet i a un poids p_i et un intérêt a_i . On est limité par le poids total P des objets à emporter et on cherche donc un sous-ensemble d'objets dont la somme des intérêts est maximale parmi ceux dont la somme des poids est inférieure ou égale à P .

i) On applique l'algorithme consistant à classer les objets par intérêt décroissant et à considérer les objets itérativement dans cette liste ; on emporte ceux dont le poids ne fait pas dépasser le poids total P acceptable. Montrer en donnant un exemple que cet algorithme ne donne pas toujours l'optimum.

ii) On applique ensuite l'algorithme consistant à classer les objets par poids croissant et à procéder ensuite comme pour l'algorithme précédent. Donner un nouvel exemple montrant que cet algorithme ne donne pas l'optimum.

iii) On applique enfin l'algorithme consistant à classer les objets suivant le rapport (intérêt/ poids) et à poursuivre comme plus haut. Montrer à nouveau que cet algorithme ne donne pas l'optimum.

2. Processeur séquentiel

On considère un ensemble de tâches T_i , $1 \leq i \leq n$, ayant toutes une durée unitaire. Pour chaque tâche sont fixés un délai d_i et une pénalité p_i qui est à acquitter quand la tâche n'est pas remplie avant le temps d_i . On veut trouver un ordre d'exécution des tâches sur une machine unique qui minimise la somme des pénalités. Soit \mathcal{F} la famille des sous-ensembles F de $\{T_1, \dots, T_n\}$ tels que $F \in \mathcal{F}$ ssi les tâches de F peuvent être toutes réalisées à temps, on appellera ces ensembles *réalisables*. Montrer que \mathcal{F} est une famille d'ensembles formant un matroïde. (On pourra chercher, étant donnés deux ensembles réalisables X et Y , chacun ordonné par délai croissant, et tels que $|X| < |Y|$, quel élément de Y on peut ajouter à X tout en conservant la réalisabilité.)

On conclura donc qu'il suffit de classer les tâches par ordre de pénalité décroissante pour obtenir la réalisation optimale.

3. Matroïde des colonnes d'une matrice

i) Étant donnée une matrice M de taille $m \times n$ à coefficients dans un corps, on note $\{C_1, \dots, C_n\}$ l'ensemble de ses colonnes. Montrer que l'on définit bien un matroïde si l'on prend comme ensembles indépendants les sous-ensembles de $\{C_i\}$ formés de colonnes linéairement indépendantes. (C'est de cet exemple que vient le mot "matroïde").

ii) On considère à présent un graphe non-orienté $G = (X, E)$. On peut lui associer une matrice d'incidence orientée A de taille $|X| \times |E|$ de la manière suivante : on étiquette chaque sommet et chaque arête puis on oriente de manière arbitraire chaque arête. Le coefficient $A_{i,j}$ est égal à 1 (resp. -1) si l'arête j est issue du sommet i (resp. pointe vers le sommet i) et à 0 sinon.

A quoi correspondent en termes de graphes les ensembles indépendants de la matrice d'incidence ainsi définie ?

4. Calculer dans un matroïde

On se donne un matroïde défini par un ensemble fini E et un algorithme qui détermine si un sous-ensemble de E est indépendant en temps polynomial en la taille du sous-ensemble.

i) Le *rang* d'un sous-ensemble X de E est le cardinal maximum d'un sous-ensemble de X qui est indépendant. Donner un algorithme qui calcule le rang de X en temps polynomial en sa taille.

ii) Une base dans un matroïde est un ensemble indépendant de taille maximale. Donner un algorithme qui construit une base contenant un ensemble indépendant X , en temps polynomial en la taille de E .

5. Un problème d'ordonnement

De nombreux problèmes rencontrés dans l'affectation de tâches sur un processeur s'apparentent à ceux concernant l'organisation du travail dans les ateliers. En voici un exemple : on suppose que l'on doit réaliser des tâches T_1, T_2, \dots, T_n sur une seule machine. Chaque tâche T_i a une durée d_i et une priorité p_i . Une réalisation de ces tâches est donnée par une permutation $T_{i_1}, T_{i_2}, \dots, T_{i_n}$ de celles-ci, représentant l'ordre dans lequel elles sont effectuées sur la machine. La date de fin F_i d'une tâche T_i est donnée par la somme des durées des tâches qui la précèdent dans la permutation augmentée de sa durée propre d_i . On mesure la *pénalité* d'une réalisation par la quantité $\sum_{i=1}^n p_i F_i$ qui devra être rendue minimum, ainsi les tâches de priorités plus grandes devraient être réalisées avant celles de priorités plus petites.

i) On suppose qu'il y a 4 tâches ayant pour durées 3, 5, 7, 4, et de priorités 6, 11, 9, 5. Laquelle de ces deux réalisations est-elle la meilleure : T_1, T_4, T_2, T_3 ou T_4, T_1, T_3, T_2 ?

ii) Soient deux tâches T_i et T_j qui se suivent dans une réalisation et telles que

$$\frac{d_i}{p_i} < \frac{d_j}{p_j}$$

laquelle de ces deux tâches doit on mettre avant l'autre pour minimiser la pénalité ?

ii) En déduire un algorithme glouton permettant d'obtenir la réalisation de plus faible pénalité.