TD d'algorithmique avancée

Corrigé du TD 7 : Algorithmes gloutons

Jean-Michel Dischler et Frédéric Vivien

Emploi du temps de salles

On suppose que l'on a un ensemble de n cours, c_1, ..., c_n, le cours c_i commençant à l'heure *début*(c_i) et finissant à l'heure *fin*(c_i). Écrivez un algorithme qui attribue une salle à chaque cours sachant que l'on veut minimiser le nombre de salles occupées et que l'on ne veut pas que deux cours aient lieu en même temps dans la même salle.

SALLES(c)
  Trier les cours par dates de début croissantes
  $n_{salles} \leftarrow 0$
  pour $i \leftarrow 1$ **à** n **faire**
    si parmi les n_{salles} utilisées il n'existe pas de salle libre à partir de l'heure *début*(c_i)
    alors $n_{salles} \leftarrow n_{salles} + 1$
      affecter c_i à la salle n_{salles}
    sinon affecter c_i à une des salles disponibles

Si l'algorithme augmente, à cause du cours c_i, le nombre de salles utilisées (n_{salles}) alors, par définition de l'algorithme, il y a cours à la date début(c_i) dans chacune des n_{salles} utilisées. En comptant c_i, il y a donc au moins $(1 + n_{salles})$ cours qui ont lieu à la date début(c_i) et il faut donc au moins $(1 + n_{salles})$ salles pour les abriter. L'algorithme est donc optimal.

Sous-graphes acycliques

Soit $G = (S, A)$ un graphe non orienté. On définit un ensemble I de sous-ensembles de A comme suit : F appartient à I si et seulement si (S, F) est un graphe acyclique.

1. Montrez que (A, I) est un matroïde.

   Pour montrer que (A, I) est un matroïde, nous devons montrer qu'il vérifie trois propriétés :

   A **est un ensemble fini.** *C'est une évidence.*

   I **est héréditaire.** *Soient H un élément de I ($H \in I$) et soit F un sous-ensemble de H ($F \subset H$). (S, F) est trivialement un graphe. De plus, c'est un sous-graphe de (S, H). Donc, si (S, F) contenait un cycle, il en irait de même de (S, H). (S, H) étant acyclique, (S, F) est acyclique et I contient F ($F \in I$).*

   Propriété d'échange. *Soient F et H deux éléments de I, $|F| < |H|$. On doit montrer qu'il existe au moins un élément x de $H \setminus F$ tel que $F \cup \{x\}$ soit élément de I.*

   Nous découpons le problème en cas :

   Sommets isolés différents. *Si H contient une arête x incidente à un sommet isolé dans (S, F), $(S, F \cup \{x\})$ est acyclique et $(F \cup \{x\}) \in I$.*

   Mêmes sommets isolés. *Les sommets isolés de F sont aussi isolés dans H. De deux choses l'une : soit F et H n'ont pas les mêmes composantes connexes, soit ils ont les mêmes.*

Composantes connexes différentes. *Soit x une arête entre deux sommets u et v appartenant à des composantes connexes différentes de (S, F). Si $(S, F \cup \{x\})$ n'est pas acyclique, alors il existait déjà une chaîne dans (S, F) reliant u et v ce qui contredit l'hypothèse que u et v appartiennent à des composantes connexes différentes de (S, F). Donc $F \cup \{x\} \in I$.*

Mêmes composantes connexes. *Les composantes connexes des deux graphes sont exactement les mêmes. Chacune de ces composantes connexes est un sous-graphe connexe acyclique de H (resp. F) et contient donc une arête de moins que de sommets (d'après le point 4 du théorème 7 de caractérisation des arbres vu en cours). Par conséquent, le nombre d'arêtes de H (resp. F) est exactement le nombre de ses sommets, $|S|$, moins le nombre de ses composantes connexes. Donc, F et H ont le même nombre d'éléments, ce qui contredit l'hypothèse : $|F| < |H|$. Ce cas est donc impossible.*

On pourrait bien sûr se contenter ne n'étudier que les cas où les composantes connexes sont les mêmes ou sont différentes. Il m'a semblé que le cheminement suivi ici était plus naturel...

2. Proposez un algorithme pour calculer un plus grand (en nombre d'arêtes) sous-graphe acyclique de G.

   Pour utiliser les résultats vus en cours, il nous faut un matroïde pondéré. Il nous manque donc juste la fonction de pondération. Il nous suffit ici d'utiliser une fonction qui associe un poids de 1 à chaque arête !

   PLUS-GRAND-SOUS-GRAPHE-ACYCLIQUE($G = (S, A)$)
   $F \leftarrow \emptyset$
   pour x appartenant à A **faire si** $(S, F \cup \{x\})$ est acyclique **alors** $F \leftarrow F \cup \{x\}$
   renvoyer (S, F)

   Tout simplement !

Ordonnancement de tâches avec priorités

On doit réaliser un ensemble de n tâches T_1, ..., T_n sur une unique machine. Chaque tâche T_i a une durée d_i et une priorité p_i. Une réalisation des tâches T_1, ..., T_n en est une permutation T_{i_1}, T_{i_2}, ..., T_{i_n} représentant l'ordre dans lequel elles sont exécutées. La date de fin d'exécution F_i d'une tâche T_i est égale à la somme des durées des tâches qui la précèdent dans la permutation plus sa durée propre (d_i). La *pénalité* d'une exécution vaut $\sum_{i=1}^{n} p_i F_i$. On cherche un ordonnancement qui minimise cette pénalité.

1. Soit quatre tâches de durées respectives 3, 5, 7 et 4, et de priorités respectives 6, 11, 9 et 5. Des deux réalisations (T_1, T_4, T_2, T_3) et (T_4, T_1, T_3, T_2), quelle est la meilleure ?

   (a) (T_1, T_4, T_2, T_3) : $F_1 = d_1 = 3$, $F_4 = d_1 + d_4 = 7$, $F_2 = d_1 + d_4 + d_2 = 12$, $F_3 = d_1 + d_4 + d_2 + d_3 = 19$. La pénalité vaut donc : $\sum_{i=1}^{4} p_i F_i = 6 \times 3 + 11 \times 12 + 9 \times 19 + 5 \times 7 = 18 + 132 + 171 + 35 = 356$.

   (b) (T_4, T_1, T_3, T_2) : $F_4 = d_4 = 4$, $F_1 = d_4 + d_1 = 7$, $F_3 = d_4 + d_1 + d_3 = 14$, $F_2 = d_4 + d_1 + d_3 + d_2 = 19$. La pénalité vaut donc : $\sum_{i=1}^{4} p_i F_i = 6 \times 7 + 11 \times 19 + 9 \times 14 + 5 \times 4 = 42 + 209 + 126 + 20 = 397$.

   La première réalisation est donc la meilleure des deux.

2. Soient deux tâches T_i et T_j consécutives dans une réalisation telles que : $\frac{p_i}{d_i} < \frac{p_j}{d_j}$. Afin de minimiser la pénalité, laquelle doit être exécutée en premier et laquelle en second ?

   On a donc deux réalisations identiques sauf que dans la première T_i est exécutée juste avant T_j et dans la deuxième T_j est exécutée juste avant T_i. Soit d la somme de la durée des tâches exécutée dans la première (resp. deuxième) réalisation avant le début de l'exécution de d_i (resp. d_j). Les pénalités des deux réalisations ne diffèrent que pour les termes relatifs à T_i et T_j :

   Première réalisation. $(d + d_i)p_i + (d + d_i + d_j)p_j = ((d + d_i)p_i + (d + d_j)p_j) + d_i p_j$.

   Deuxième réalisation. $(d + d_j)p_j + (d + d_j + d_i)p_i = ((d + d_j)p_j + (d + d_i)p_i) + d_j p_i$.

   Comme $\frac{p_i}{d_i} < \frac{p_j}{d_j}$, $d_j p_i < p_j d_i$ et la deuxième réalisation est moins coûteuse que la première. Par conséquent, T_j doit être exécutée avant T_i.

3. En déduire un algorithme.

   Dans une solution optimale on doit donc forcément avoir $\frac{p_{i+1}}{d_{i+1}} < \frac{p_i}{d_i}$*, ce qui ne nous laisse guère le choix quant à la solution.*

   ORDO-AVEC-PRIORITÉS(T, d, p)

   Exécuter les tâches par ratio $\frac{p_i}{d_i}$ décroissant