Majeure Informatique Promotion 2004

Cours de Conception et analyse d'algorithmes
Corrigé de l'examen du mercredi 28 mars 2007

I. Problèmes NP-complets et algorithmes d'approximation *(7 points).*

L'objet de ce problème est d'étudier un problème d'ordonnancement très simple : il y a n travaux à réaliser sur m machines ; chaque travail A_i nécessite un temps p_i et doit être effectué sur une seule machine sans interruption. On cherche à déterminer une affectation des travaux sur les machines qui minimise le temps de la réalisation de l'ensemble des travaux.

1. Version *décision* du problème, celle-ci est NP-complète.

On se donne un ensemble de n travaux de durées p_i, $(1 \leq i \leq n)$ à exécuter sur m machines et un nombre t, il s'agit alors de savoir s'il existe une affectation des travaux sur les machines de façon telle que pour chaque machine la somme des temps des travaux qui lui sont affectés soit inférieure à t.

On va montrer que le problème PARTITION *se réduit à ce problème d'affectation. En effet soit $a_1, a_2, \ldots, a_n$ une instance de partition, il s'agit alors de savoir si on peut partager les a_i en deux sous-ensembles de sommes égales. A cette donnée pour* PARTITION *on associe la donnée suivante pour notre probleme d'affectation on suppose qu'il y a deux machines et que les a_i sont des durées de travaux, on prend comme temps t la moitié de la somme des a_i. On voit alors que l'on peut partager les a_i en deux parties de sommes égales si et seulement si on peut réaliser tous les travaux en temps inférieur ou égal à t.* □

2. Exprimer ce problème sous la forme de l'optimisation, sous contraintes, d'une fonction à nm variables entières.

On note $x_{i,j}$ des variables telles que $1 \leq i \leq n$ et $1 \leq j \leq m$, la signification de $x_{i,j} = 1$ sera que le travail A_i est affecté à la machine j. On doit alors avoir $x_{i,j} \in \{0, 1\}$ et pour tout $1 \leq i \leq n$

$$\sum_{j=1}^{m} x_{i,j} \quad = \quad 1$$

qui signifie que chaque travail doit être reálisé sur l'une des machines et il s'agit de minimiser :

$$\max_{j=1,\ldots,m} \sum_{i=1}^{n} x_{i,j} p_i$$

□

3. Soit T_{opt} le temps obtenu pour l'affectation optimale, montrer que :

$$mT_{opt} \geq \sum_{i=1}^{n} p_i$$

Dans l'affectation optimale chaque machine fonctionne un temps inférieur ou égal à T_{opt} ; le temps total de fonctionnement de l'ensemble des machines est inférieur ou égal à mT_{opt}, or ce temps ne saurait être inférieur à la somme des temps nécessaire à chacun des travaux. □

On considère l'algorithme glouton qui classe de façon quelconque tous les travaux ; ensuite, à chaque fois qu'une machine est disponible on lui affecte le premier travail en attente. Soit A_k le travail qui se termine en dernier dans l'algoritme glouton et soit t_k l'heure du début de l'exécution de ce travail.

4. Montrer que :

$$t_k \quad \leq \quad \frac{1}{m} \sum_{i \neq k} p_i$$

En déduire une majoration du temps T_g obtenu par l'algorithme glouton.

Jusqu'à l'instant t_k toutes les m machines fonctionnent en continu, le temps total passé est donc mt_k, l'ensemble formé de travaux $A_i, i \neq k$ ne sera pas terminé avant cet instant (sinon A_k aurait commencé plus tôt) ainsi

$$mt_k \quad \leq \quad \sum_{i \neq k} p_i$$

d'où le résultat demandé.

Le temps obtenu par l'algorithme glouton est égal à $t_k + p_k$ puisque c'est la dae de fin du travail A_k qui est terminé en dernier. On a donc $T_g \leq p_k + \frac{1}{m} \sum_{i \neq k} p_i$
□

5. En déduire que l'algorithme glouton est $\frac{m-1}{2m-1}$-approché.

Le majorant obtenu pour T_g peut s'écrire :

$$T_g \leq \frac{m-1}{m} p_k + \frac{1}{m} \sum_{i=1}^{n} p_i$$

mais on a aussi $p_k \leq T_{opt}$ d'où d'après la Question 3 :

$$T_g \leq (\frac{m-1}{m} + 1) T_{opt}$$

ce qui donne

$$T_g - T_{opt} \leq T_g - \frac{m}{2m-1} T_g$$

et

$$\frac{T_g - T_{opt}}{T_g} \leq 1 - \frac{m}{2m-1} \quad = \quad \frac{m-1}{2m-1}$$

ce qui donne le résultat. □

II. Un algorithme probabiliste pour compter *(6 points).*

On considère le problème COMPTER(F) pour lequel la donnée est une formule booléenne F en forme disjonctive, c'est à dire qu'elle se présente sous la forme d'un *ou* de plusieurs monômes F_i, $(F = F_1 \vee F_2 \vee \cdots \vee F_p)$; où chaque F_i est le *et* de plusieurs variables (complémentées ou pas) $(F_i = y_1 \wedge y_2 \wedge \cdots \wedge y_{q_i})$.

Il s'agit alors de déterminer le nombre $c(F)$ de valeurs des variables $x_1, x_2, \ldots x_n$ qui donnent la valeur *true* à F.

1. Déterminer $c(F)$ pour la fonction à 4 variables :

$$F = (x_1 \wedge \overline{x}_3) \ \vee \ (\overline{x}_2 \wedge \overline{x}_3) \ \vee \ (x_1 \wedge x_2 \wedge x_4)$$

Le premier monôme est satisfait par 4 valeurs des variables, le suivant ajoute deux autres valeurs aux précédentes et le dernier une seule, au total $c(F) = 7$.
□

2. On applique une méthode aléatoire qui consiste à tirer au hasard avec probabilité $\frac{1}{2}$ la valeur (*true* ou $false$) de chacune des n variables et de déterminer si l'affectation trouvée donne la valeur *true* à F. Le tirage est effectué m fois et le nombre de fois où la valeur *true* est obtenue pour F est m_1, quelle est alors une approximation de $c(F)$?

On suppose que le pourcentage du nombre de solutions dans l'ensemble des 2^n *affectations est le même que sur toutes celles qui ont été tirées au hasard. On peut alors prendre comme valeur approchée du nombre de solutions :*

$$\frac{m_1 2^n}{m}$$

□

Nouvel algorithme :
- On tire au hasard un nombre a entre 1 et $\sum_{i=1}^{p} u_i$, et on détermine j tel que

$$\sum_{i=1}^{j-1} u_i < a \leq \sum_{i=1}^{j} u_i$$

- On tire au hasard une affectation de valeurs qui donnent la valeur *true* à F_j et on cherche si cette affectation donne aussi la valeur *true* à un F_i tel que $i < j$; si tel n'est pas le cas on incrémente le compteur du nombre de tirages réussis.
- Après m tirages on obtient $m_1 \leq m$ tirages réussis, il s'agit alors d'obtenir une approximation de $c(F)$

3.
- **3.1** On note U_i l'ensemble des affectations qui donnent la valeur *true* à F_i, quelle est le nombre d'élements u_i de U_i si F_i contient q_i variables sur un total de n ?

On peut chosir de façon quelconque la valeur des $n - q_i$ variables qui n'apparaissent pas dans F_i soit :

$$u_i = 2^{n-q_i}$$

□

– **3.2** On note U l'ensemble des couples (i, α) tels que i est un nombre compris entre 1 et p et α une affectation qui donne la valeur *true* à F_i. Quel est le cardinal de U en fonction des q_i de p et de n ?

Il faut faire la somme des p valeurs obtenues soit

$$|U| \quad = \quad \sum_{i=1}^{p} 2^{n-q_i}$$

□

– **3.3** Soit V le sous ensemble de U formé des (i, α) tels que α est une affectation des valeurs des variables qui donnent la valeur $false$ à tous les F_j pour $j < i$. Exprimer à l'aide de V la valeur de $c(F)$

On ne doit compter qu'une seule fois chaque affectation α qui donne la valeur true, si on l'a comptée pour F_i et que F_j $(j > i)$ prend aussi la valeur true, il ne faut pas la compter une deuxième fois. Mais c'est exactement ce qui se passe en considérant l'ensemble V, on a donc :

$$c(F) \quad = \quad |V|$$

□

– **3.4** Donner une approximation de $c(F)$ à l'aide des nombres m de tirages et m_1 de tirages réussis donnés par l'algorithme.

Comme pour la question précédente on suppose que la repartition de l'echantillon est le même que celui de l'ensemble complet, l'approximation est alors

$$\frac{m_1(\sum_{i=1}^{p} 2^{n-q_i})}{m}$$

□

– **3.5** Cette méthode donne-t-elle à votre avis une meilleure approximation de $c(F)$ que celle de la question **2** ? Expliquer pourquoi.

Dans le deuxième algorithme le nombre de réussites dans l'échantillon sera beaucoup plus grand que dans le premier cas, puisque 2^n est largement supérieur à $\sum_{i=1}^{p} 2^{n-q_i}$. Dans le premier cas on multiplie un petit nombre approché par un grand nombre exact, l'erreur sera plus grande que si l'on multiplie un plus grand nombre approché par un plus petit nombre exact.

□

III. Arbres recouvrants dynamiques *(7 points).*

Soit un graphe $G = (X, E, w)$ non orienté dont l'ensemble de sommets est $1, 2, \ldots, n$ et dont l'ensemble des arêtes $E = \{e_1, e_2, \ldots e_m\}$ est muni d'un poids w (une application de E dans l'ensemble des rationnels positifs) tel que tous les poids sont différents. On adopte la convention :

$$0 < w(e_1) < w(e_2) < \ldots < w(e_{m-1}) < w(e_m)$$

Dans ce cas il existe *un et un seul arbre recouvrant de poids minimum,* noté T que l'on supposera calculé.

On considère dans cet exercice la modification de la valeur du poids d'une des arêtes du graphe ; pour les deux premières questions cette arête n'appartiendra pas à T et pour les deux questions suivantes, ce sera un élément de T.

1. Algorithme sans justification qui calcule le nouvel arbre minimal lorsque l'on diminue d'une quantité $\alpha > 0$ le poids d'une seule arête $e_k = \{a, b\}$ qui n'est pas dans T. On suppose que l'arbre T est donné par une racine et un tableau `pere`, tel que pour chaque sommet i, `pere[i]` est le premier sommet rencontré en remontant de i à la racine.

On ajoute l'arête e_k à T cela crée un cycle et un seul on supprime ensuite de ce cycle l'arête ayant le plus grand poids pour w'. On obtient ainsi un arbre T', qui peut être égal à T si e_k est encore l'arête de plus grand poids après l'avoir lestée de α.

L'algorithme doit d'abord rechercher le cycle qui contient e_k ceci se fait en remontant dans le tableau `pere` *des deux extremités de e_k jusqu'à retrouver un sommet d'une des remontée au cours de l'autre. Ceci se fait en temps $O(\ell)$. Ensuite on recherche l'arête de plus grand poids dans ce cycle et on la supprime ; là encore la complexité est en $O(\ell)$.*

□

2. Jusifier la validité de l'algorithme que vous avez proposé pour répondre à la question précédente. Pour cela vous considèrerez le nouveau poids w' ($w'(e) = w(e)$ si $e \neq e_k$ et $w'(e_k) = w(e_k) - \alpha$), l'indice j tel que $w(e_j) < w'(e_k) < w(e_{j+1})$ et $F_1, F_2, \ldots, F_p$ l'ensemble des composantes connexes de $T \cap \{e_1, e_2, \ldots, e_j\}$. Vous comparerez les deux exécutions de l'algorithme de Kruskal : celle avec le poids w et celle avec le poids w'.

L'ordre dans lequel on considère les arêtes par l'algorithme de Kruskal avec le poids w' est très voisin de celui pour le poids w, sauf que e_k est passé du rang k au rang $j+1$. Ainsi jusqu'à la considération de l'arête e_j le déroulement de l'algorithme est exctement le même dans le deux cas. Lorsque l'on considère l'arête e_k pour w' il y a deux situations à examiner :

- *Cette arête crée un cyle avec des arêtes de T toutes d'indices inférieurs ou égaux à j. Dans ce cas elle est l'arête la plus lourde du cycle considéré plus haut et elle ne sera pas retenue par l'algorithme de Kruskal. Les deux arbres de poids minimaux seront égaux.*

– *Cette arête ne crée pas de cycle, elle est alors conservée dans l'arbre T' de poids minimal pour w' alors qu'elle ne l'était pas pour T. Les arêtes qui ont été rejetées ensuite par l'algorithme de Kruskal pour w le seront aussi pour w', et les arêtes qui ont été retenues le seront aussi, sauf celle qui crée un cycle avec e_k et T et ce sera nécessairement l'arête de plus grand poids de ce cycle.*

□

3. On considère maintenant le cas où on augmente de $\alpha > 0$ le poids d'une arête $e_k = \{a, b\}$ de T, proposer un algorithme en $O(m)$ qui calcule le nouvel arbre de poids minimum.

La suppression de l'arête e_k de l'arbre T partage celui-ci en deux composantea connexes ayant pour ensembles de sommets X_1 et X_2 . On considère alors l'ensemble U des arêtes de G qui relient un sommet appartenant à X_1 à un sommet appartenant à X_2 et on cherche l'arête e de plus petit poids (pour w') appartenant à U. L'arbre de poids minimal est alors $T - e_k + e$. □

4. Justifier la validité de votre algorithme par un raisonnement du même type que celui qui a été fait pour la Question 2.

L'ordre dans lequel on considère les arêtes par l'algorithme de Kruskal avec le poids w' est très voisin de celui pour le poids w, sauf que e_k est passé du rang k à un rang plus élevé. Ainsi avant la considération de l'arête e_k pour w ; le déroulement de l'algorithme est exactement le même dans le deux cas. On note alors $F_1, F_2, \ldots, F_p$ l'ensemble des composantes connexes de $T \cap \{e_1, e_2, \ldots, e_{k-1}\}$.
Lorsque l'on considère les arêtes pour w' celles-ci sont acceptées lorsqu'elles ont été acceptées pour w ; une de plus le sera : la plus légère parmi celles reliant les composantes connexes F_i et F_j qui étaient reliées par e_k dans T'. □

5. L'hypothèse suivant laquelle tous les poids des arêtes de G sont différents, est-elle nécessaire pour la vailidité des algorithmes que vous avez proposés ?

Cette hypothèse n'est pas nécessaire en effet on peut s'y ramener une fois que l'on a déterminer l'ordre total de parcours des arêtes au début de l'algorithme de Kruskal. Pour une suite d'arêtes $e_i, e_{i+1}, \ldots e_j$ de même poids (telle que $w(e_{j+1}) > w(e_j)$ on considère la quantité $\varepsilon = \frac{w(e_{j+1}) - w(e_j)}{m}$ et on ajoute $k\varepsilon$ à e_{i+k}. Toutes les arêtes ont alors des poids différents et l'algorithme de Kruskal se comporte de la même façon dans les deux cas. □