Master Informatique Fondamentale (1ère année)
Algorithmes d'approximation — **TD7**

PTAS et sacs-à-dos

3 décembre 2004

Exercice 1 : Bin-packing.

Dans le but de construire un PTAS pour le problème du bin-packing (empaquetage), on commence par étudier la restriction suivante du problème.

Question 1. *Pour tout $\epsilon > 0$ et tout entier $K > 0$ fixés, démontrer qu'il existe un algorithme polynomial (en n, le nombre d'objets) qui résout exactement la restriction du problème aux instances où tous les objets sont de taille $\geq \epsilon$ et où il n'existe qu'au plus K tailles d'objets différentes.*

Réponse : Si tout les objets ont une taille supérieure à ϵ, on peut mettre au plus $1/\epsilon$ objets dans une boite. Le rangement d'une boite est donc décrit par un $1/\epsilon$-uplet donnant les tailles des objets rangés. Ces tailles sont choisis parmi k valeurs et éventuellement 0 (pour dire qu'on ne met pas d'objet), il y a donc au plus $(k+1)^{1/\epsilon}$ rangements possibles pour une boite.

Comme il n'y a aucun intéret à garder des boites vides, toute type de boites a au moins un objet, donc, dans une solution, il ne peut avoir au maximum n fois le même type de boite. On peut ainsi décrire une solution par un $(k+1)^{1/\epsilon}$-uplet de valeurs dans $\{0..n\}$ (le nombre de fois qu'apparait chaque type de boite) : il y a donc au plus $(n+1)^{(k+1)^{1/\epsilon}}$ solutions possibles.

On peut ainsi réaliser un parcours polynomial des solutions et trouver l'optimal.

□

On appelle *First-fit* l'algorithme qui prend les objets dans l'ordre et les place dans la première boîte ouverte (prise dans l'ordre de première ouverture) dans laquelle l'objet rentre et qui ouvre une nouvelle boîte si aucune boîte ne convient.

Question 2. *Soient $\epsilon > 0$ fixé et $S = \langle x_1, \ldots, x_n \rangle$ une instance du bin-packing. On note $S^+ = \langle x_i : x_i \geq \epsilon \rangle$ et $S^- = \langle x_i : x_i < \epsilon \rangle$ les ensembles des objets de taille $\geq \epsilon$ et $< \epsilon$ respectivement. Démontrer que si on a un empaquetage B des objets de S^+ dans b boîtes, alors l'algorithme qui place les objets de S^- dans l'empaquetage B avec l'algorithme First-Fit, construit un empaquetage B' des objets dans b' boîtes avec :*

$$b' \leq \max(b, \frac{\mathrm{OPT}(S)}{1-\epsilon} + 1).$$

Réponse : Montrons que $b' \leqslant max(b, \frac{OPT(S)}{1-\epsilon} + 1)$:

- Si l'ajout des petits éléments n'a pas ouvert de nouvelle boite alors $b = b'$ et la proposition est vrai.
- Si l'ajout des petits éléments a ouvert de nouvelles boites, le passage en `First-fit` a notamment ouvert la dernière : Les objets de cette dernière boite ne peuvent pas entrer dans les $b' - 1$ premières boites, autrement dit les $b' - 1$ premières boites sont au moins remplies à $1 - \epsilon$. Ainsi on a $Volume(S) \geqslant (b' - 1)(1 - \epsilon)$ et comme le volume est un minorant de la solution optimale, $OPT(S) \geqslant Volume(S)$, on obtient :

$$b' \leqslant \frac{OPT(S)}{1-\epsilon} + 1$$

On a donc $b' \leqslant max(b, \frac{OPT(S)}{1-\epsilon} + 1)$ □

Exercice 2 : NP-difficulté forte et PTAS.

Un algorithme est pseudo-polynomial pour un problème Π si son temps de calcul sur chaque instance I est polynomial en $|I|_u$ (la taille de l'instance quand tous les nombres sont écrits en unaire).

Un problème Π_0 est NP-difficile *au sens fort* si sa restriction aux instances I telle que $|I|_u$ est polynomial en $|I|$ est un problème NP-complet.

Question 3. *Montrer que l'existence d'un algorithme pseudo-polynomial pour un problème NP-difficile au sens fort implique P=NP.*

Réponse : Soit Π un problème NP-difficile au sens fort.
Soit Π_0 sa restriction aux instances I telles que $|I|_u = P(|I|)$. Alors Π_0 est NP-complet.
Soit A un algorithme pseudo-polynomial pour Π, alors A_{Π_0} est polynomial en $|I|$ et comme Π_0 est NP-complet, alors $P = NP$. □

Question 4. *Donner un exemple de problème NP-difficile qui n'est pas NP-difficile au sens fort (sauf si P=NP). Donner un exemple de problème NP-difficile au sens fort.*
Indication. On pourra considérer PARTITION et TSP respectivement.

Réponse :

1. Montrons que partition n'est pas un problème NP-difficile au sens fort.
   On va pour cela exhiber un algorithme qui résoud ce problème en temps pseudo-polynomial :

   - $L \leftarrow (0,0)$
   - Pour chaque objet de poids i,
     - $L' \leftarrow L$
     - Pour chaque $(a,b) \in L$
       - $L' \leftarrow L' U \{(a+i,b),(a,b+i)\}$
     - $L \leftarrow L'$

   Si $(s/2, s/2) \in L$, alors une partition existe, sinon il n'en existe pas.
   On remarque ici que $a + i \leq s/2$ et que $b + i \leq s/2$. Ainsi l'algorithme est bien polynomial en $s/2$
2. Montrons que TSP est NP-dificile au sens fort :
   Soit $A = (E, V)$ une instance de circuit Hamiltonien. On cré une instance de TSP en prenant le graphe A que l'on complète : on met un poid 2 sur les arêtes de V et 1 sur les autres.
   On a alors une instance de TSP dont l'écriture en unaire est polynomiale en la taille de l'instance. Donc si on a une solution pseudo polynomiale à ce problème, on a une solution à circuit Hamiltonien en temps polynomial. TSP est donc NP-complet au sens fort.

□

Question 5. *Soit Π un problème de minimisation dont la fonction objective f_Π ne prend que des valeurs entières et telle qu'il existe un polynôme p vérifiant pour toute instance I : $f_\pi\big(\mathrm{OPT}(I)\big) \leq p(|I|_u)$. Montrer que si Π admet un FPTAS, alors Π admet un algorithme pseudo-polynomial. Montrer que si Π est* en plus *NP-difficile au sens fort, alors P=NP.*

Réponse : Soit Π un problème de minimisation tel que f_Π, sa fonction objective soit à valeurs dans $\mathbb{N}$. On suppose de plus que $f_\Pi(OPT(I) \leq p(|I|_u)$.
Supposons que Π admette un $FPTAS$, alors $\forall \epsilon \geq 0, \exists A_\epsilon$, un algorithme polynomial en n tel que
$f_\Pi(A_\epsilon(I)) \leq (1+\epsilon)OPT(I)$
$f_\Pi(A_\epsilon(I)) \leq (1+\epsilon)p(|I|_u)$
donc, en prenant $\epsilon = \frac{1}{p(|I|_u)}$ on a un algorithme polynomial en $|I|_u$ qui résoud Pi (car f_Π est à valeurs entières). Donc Π admet une algorithme pseudo-polynomial.
D'après la question 3, si de plus Π est NP-difficile, $P = NP$.

□

Exercice 3 : Patchons le glouton du sac-à-dos.

On étudie l'algorithme glouton A qui trie les objets par ratio valeur / taille décroissant et en sélectionne le plus possible dans cet ordre de manière gloutonne.

Question 6. *Donner une famille d'instances sur laquelle l'algorithme glouton produit des résultats arbitrairement mauvais.*

Réponse : L'idée intuitive pour faire planter l'algorithme est de considèrer une toute petite pépite d'or et un gros bloc d'argent : notre algorithme prendra l'or alors que prendre l'argent aurait été plus profitable. L'instance critique correspondante est donc composée d'un objet de taille 1 et de valeur 1 (l'or) ainsi que d'un objet de taille n et de valeur $n-1$ (l'argent). □

Question 7. *Pour une instance I, on trie les objets par ordre décroissant de rentabilité et on note o_j le premier objet non sélectionné par l'algorithme glouton. Montrer que :*

$$val\big(\mathrm{OPT}(I)\big) \leq val\big(A(I)\big) + val(o_{j+1}).$$

Réponse : On considère les sacs à dos de l'algorithme glouton et d'une solution optimale et on y ordonne les objet par ordre décroissant de densité ($\frac{valeur}{taille}$) : au fond du sac (abscisse 0) les densités sont plus grandes qu'en haut du sac (abscisse T). Soit $t = \sum_{i \leq j} t(o_i)$ (la longueur des objets sélectionnés par le glouton avant le premier refus) et soit x un point du sac à dos dans l'intervalle $[0, t]$. Par choix glouton, $dg(x) \geq dO(x)$ (densité glouton, densité OPT). Par ailleurs, si $x \in [t, T]$ alors $dO(x) \leq d(o_{j+1})$. On a donc

$$\begin{aligned} v(OPT) = \int_{x=0}^{T} dO(x)dx &\leq \int_{x=0}^{t} dO(x)dx + \int_{x=t}^{T} d0(x)dx \\ &\leq \int_{x=0}^{t} dg(x)dx + \int_{x=t}^{T} d(o_{j+1})dx \\ &\leq v(glouton) + d(o_{j+1})(T-t) \end{aligned}$$

Or $T - t \leq t(o_{j+1})$ car l'algorithme glouton a refusé l'objet o_{j+1}, donc $v(OPT) \leq v(glouton) + v(o_{j+1})$. □

Question 8. *En déduire une 1/2-approximation "presque" gloutonne.*

Réponse : On exécute l'algorithme glouton et on renvoie le maximum entre la solution gloutonne et le premier objet non sélectionné. Ce maximum est supérieur à la moyenne des deux donc à la moitié de l'optimal. □

Exercice 4 : Débordement de coût minimum.

Considérons le problème suivant : étant donnés n objets de tailles positives $t_1, \ldots, t_n$, et de coûts positifs $c_1, \ldots, c_n$, et une constante W, trouver un sous-ensemble $S \subset \{1, \ldots, n\}$ de coût minimum tel que $\sum_{i \in S} t_i \geq W$.

Question 9. *Proposez une réduction polynomiale du problème Partition, NP-difficile, démontrant que le problème du débordement minimum est NP-difficile.*

Rappel. Étant donné n entiers positifs $x_1, \ldots, x_n$, le *problème Partition* consiste à déterminer s'il existe un ensemble $S \subset \{1, \ldots n\}$ tel que : $\sum_{i \in S} x_i = \sum_{i \notin S} x_i$.

Réponse : Soit I = (x_1, ..., x_n) une instance de 2-Partition. On pose $\forall i \in [1, n]$, $c_i = t_i = x_i$ et $W = \frac{1}{2}\sum_{i=1}^{n} x_i$ (I'). I est 2-partitionnable ssi I' admet un débordement de coût $\leq$ W. En effet :

" $\Rightarrow$ " $\exists S \subseteq \{1, ..., n\}, \sum_{i \in S} x_i = \frac{1}{2}\sum_{i=1}^{n} x_i = W$
S est un débordement, c.à.d. $\sum_{i \in S} t_i = W \geq W$, de coût $\leq$ W ($\sum_{i \in S} c_i = W \leq W$) pour I !

" $\Leftarrow$ " S avec $W \leq \sum_{i \in S} c_i = \sum_{i \in S} c_i \leq W$

□

On étudie l'algorithme glouton suivant :

1. trier les objets par rapport coût sur taille croissant, quite à renuméroter : $c_1/t_1 \leq \ldots \leq c_n/t_n$;
2. prendre i, le plus petit indice tel que $\sum_{j \leq i} t_j \geq W$, et poser $S = \{1, \ldots, i\}$.

Question 10. *Démontrer que cet algorithme peut produire des solutions arbitrairement mauvaises, même si tous les objets sont de taille $< W$.*

Réponse : On prend 3 sortes des objets :

- objet1 de taille $t = W - \epsilon$ avec coût $c = W - \epsilon$
- objet2 de taille $t = W$ avec coût $c = k * W$
- objet3 de taille $t = \epsilon$ avec coût $c = (k+1) * \epsilon$

$T = \sum t_i$, $\epsilon = \frac{1}{k}$
Le coût de l'algorithme donne $c(A) = (k+1)W - \epsilon =: a$, parce qu'il ne prend que les objets de sorte 1 et 2. Le coût de l'OPT donne $c(OPT) = W + k\epsilon =: b$, parce qu'il ne prend que les objets de sorte 1 et 3. Le rapport de a et b est $\frac{a}{b} = O(k)$, donc il est arbitrairement grand. □

Question 11. *Poser $C = \max_i c_i$. Donner un algorithme pseudo-polynomial pour résoudre le problème du débordement minimum.*

Réponse : $C = max_i c_i$. On pose $S_{i,c}$ le sous-ensemble de $\{1, ..., i\}$ de coût c et de taille maximum ($T_{i,c}$).

$$T_{i,c} = max(T_{i-1,c}, t_i + T_{i-1,c-c_i})$$

$$T_{1,c} = \begin{cases} t_1 & \text{si } c_1 = c, \\ 0 & \text{sinon.} \end{cases}$$

La solution est le plus petit c tel que $t_{n,c} \geq W$.
On calcule au plus n * nC éléments du tableau, car nC est un majorant du coût de n importe quel instance. □

Question 12. *Proposer un schéma d'approximation totalement polynomial pour le problème du débordement minimum, et prouver-le.*

Indication. Commencer par supposer que $C \leq \text{OPT}$.

Réponse : $C = max_i c_i$. On pose $K = \frac{\epsilon C}{n}$. On transforme le coût des objets de l'instance I en coût des objets de l'instance I' avec $c_i \longrightarrow c'_i = \lceil \frac{C_i}{K} \rceil$ et $t_i \longrightarrow t'_i$ pour contrôler le coût. On fait la supposition suivante :

$$C \leq OPT(I) \tag{1}$$

Avec la programmation dynamique sur I', comme $\max c'_i \leq O(\frac{n}{\epsilon})$, on trouve $Opt(I')$ en temps polynômial (en les deux variables n et $\frac{1}{\epsilon}$). On cherche une solution pour I :
$\forall i, c'_i = \lceil \frac{c_i}{K} \rceil$ donc $c'_i - 1 < \frac{c_i}{K} \leq c'_i < \frac{c_i}{K} + 1$
Soit S optimal pour I :

$$\sum_{i \in S} c_i \geq \sum_{i \in S} (c'_i K - K) \geq K \sum_{i \in S} c'_i - nK$$

Soit S' optimal pour I' :

$$\sum_{i \in S} c_i \geq K \sum_{i \in S} c'_i - nK$$

$$OPT(I) \geq K * OPT(I') - \epsilon C$$

or

$$\sum_{i \in S'} c_i \leq \sum_{i \in S'} K c'_i = K * OPT(I')$$

donc $c(S') \leq K * OPT(I') \leq OPT(I) + \epsilon C \leq (1+\epsilon) OPT(I)$ sous l'hypothèse que C ≤ OPT(I).
Pour le cas général :

> trier les objets par ordre de coût $c_1 \leq ... \leq c_n$
> pour i de 1 à n : soit S_i la solution approchée par $[1, i]$
> retourner le S_i minimisant $c(S_i$ pour i $\in [1, n]$

Preuve :
$\exists i, C_[i] \leq OPT(I)$ et $OPT(I) \subseteq [1, i]$ alors $S_i \leq (1+\epsilon) * OPT(I)$ car $OPT(I) = OPT(I_i)$ □

♣