TD - Algorithmes d'approximation

Algorithmique Avancée – ENSIMAG 2A

1 Sac à dos binaire

Soient n objets: pour $i = 1, \dots n$, l'objet i a un volume v_i et apporte un gain g_i. On veut choisir certains de ces objets pour les mettre dans un sac de capacité volumique maximale C, avec pour objectif de maximiser le gain du sac, c'est à dire la somme des gains des objets mis dans le sac. Remarque: on verra qu'il n'existe pas d'algorithme polynomial pour résoudre ce problème, sauf si P=NP.

1. On considère l'algorithme glouton suivant. On parcourt les objets par gain volumique décroissant: si l'objet courant à un volume inférieur au volume libre restant dans le sac, alors on l'ajoute au sac.
   Donner un exemple avec deux objets où cet algorithme glouton n'est pas optimal.
   Montrer que, pour tout $\epsilon > 0$, il existe une instance pour laquelle l'algorithme glouton donne une solution de valeur inférieure à $\epsilon.\bar{\phi}$ où $\bar{\phi}$ est la valeur optimale.

   Solution: On considère un sac de volume V avec deux objets: l'un de volume V et de gain V (donc gain volumique égal à 1); l'autre de volume 1 et de gain 2 (donc de gain volumique 2). La valeur de la solution optimale est $\bar{\phi} = V$. La valeur de la solution avec l'algorithme glouton est $\phi = 2$. On a donc $\phi \leq \frac{2}{V}\bar{\phi}$; en prenant $V \geq \frac{2}{\epsilon}$, on a donc $\phi \leq \epsilon.\bar{\phi}$.

2. On suppose toujours les objets triés par gain volumique décroissant, et tous de volume $v_i \leq C$. Soit s le plus grand entier tel que $\sum_{i=1}^{s} v_i \leq V$. On considère l'algorithme qui fournit en sortie la solution de valeur $\max\{\sum_{i=1}^{s} g_i; g_{s+1}\}$: donc soit les objets $\{1, \dots, s\}$, soit l'objet $s+1$ seul. Montrer que cet algorithme est une approximation à un facteur au plus 2 de l'optimal.

   Solution: Soit $v = \sum_{i=1}^{s} v_i$ le volume pris par les s premiers objets. Comme les objets sont triés par gain volumique décroissant, on a $\bar{\phi} \leq \sum_{i=1}^{s} g_i + (V - v)\frac{g_{s+1}}{v_{s+1}} = \sum_{i=1}^{s} g_i + \frac{V-v}{v_{s+1}} g_{s+1} < \sum_{i=1}^{s} g_i + g_{s+1}$. Or $\phi = \max(\sum_{i=1}^{s} g_i + g_{s+1})$ (par définition) donc $\sum_{i=1}^{s} g_i \leq \phi$ et $g_{s+1} \leq \phi$. On on déduit : $\sum_{i=1}^{s} g_i + g_{s+1} < 2\phi$.
   Donc l'algorithme est une approximation à un facteur au plus 2 de l'optimal.

3. Majorer l'écart absolu entre la solution fournie et l'optimal: en déduire une approximation plus fine qu'un facteur 2, le facteur d'approximation $c_x < 2$ dépendant de l'instance x en entrée.

   Solution: D'après la question précédente, on a $\bar{\phi} \leq \sum_{i=1}^{s} g_i + \frac{V - \sum_{i=1}^{s} v_i}{v_{s+1}} g_{s+1}$. On distingue 2 cas:

- si $\phi = \max\{\sum_{i=1}^{s} g_i; g_{s+1}\} = \sum_{i=1}^{s} g_i$: alors $\bar{\phi} - \phi \leq \frac{V - \sum_{i=1}^{s} v_i}{v_{s+1}} g_{s+1} < \bar{\phi}$;
- sinon $\phi = \max\{\sum_{i=1}^{s} g_i; g_{s+1}\} = g_{s+1}$: alors $\bar{\phi} - \phi \leq \sum_{i=1}^{s} g_i + \left(\frac{V - \sum_{i=1}^{s} v_i}{v_{s+1}} - 1\right) g_{s+1} < \bar{\phi}$.

Soit $\rho = \max\left\{\frac{V - \sum_{i=1}^{s} v_i}{v_{s+1}} g_{s+1}; \sum_{i=1}^{s} g_i + \left(1 - \frac{V - \sum_{i=1}^{s} v_i}{v_{s+1}}\right) g_{s+1}\right\}$. On a donc: $\bar{\phi} - \phi \leq \rho < \bar{\phi}$. D'où $\bar{\phi} < \left(1 + \frac{\rho}{\phi}\right)\phi$ et la valeur $c = 1 + \frac{\rho}{\phi}$, qui dépend de l'entrée est strictement inférieure à 2. On en déduit que, pour chaque instance x en entrée, l'algorithme fournit une approximation à un facteur au plus $c_x < 2$ de l'optimal.

2 Bin-packing et first-fit

Soient n objets de taille s_i avec $0 < s_i < 1$. On veut ranger ces n objets dans des sacs de taille 1, en utilisant un nombre minimum de sacs. Sot $C^\star$ le nombre minimal de sacs avec un rangement optimal.
L'algorithme glouton "first-fit" consiste à prendre les objets 1 par 1 et mettre un objet dès qu'on trouve un sac dans lequel il rentre. Soit C le nombre de sacs nécessaires pour first-fit. L'ordre de parcours des sacs (par exemple si l'on parcourt le sac que l'on vient d'ajouter en premier ou en dernier) peut avoir une influence sur la performance.

1. Montrer qu'un minorant de $C^\star$ est : $C^\star \geq \lceil \sum_{i=1}^{n} s_i \rceil$.

Solution: Il faut au moins autant de sacs que la taille totale à stocker: $C^\star \geq \sum_{i=1}^{n} s_i$. Comme $C^\star$ est entier, on en déduit: $C^\star \geq \lceil \sum_{i=1} s_i \rceil$.
Le minorant est atteint par exemple si $s_i = 0.5 \forall i$.

2. Montrer que l'heuristique first-fit laisse au plus un sac à moitié vide au moins. En déduire que l'entier C vérifie $C \leq \lceil 2. \sum_{i=1} s_i \rceil$.

Solution: Supposons que deux sacs S_1 et S_2 sont à moitié vides, avec S_1 créé avant S_2. Alors, First-Fit aurait du mettre tous les objets de S_2 dans S_1: c'est absurde puisque First-Fit ne prend un nouveau sac S_2 que lorsqu'aucun objet ne rentre dans le sac S_1 déjà existant.
ll y a donc soit C, soit $C - 1$ sacs au moins à moitié pleins :

- si il y en a C alors $C \leq 2. \sum_{i=1}^{n} s_i \leq \lceil 2. \sum_{i=1} s_i \rceil$.
- si il en a $C - 1$: soit $I \subset \{1, \ldots, n\}$ les indices des objets qui sont dans le sac à moitié vide alors $C - 1 \leq 2. \sum_{i=1, i \notin I}^{n} s_i < 2. \sum_{i=1}^{n} s_i$. D'où, comme C est entier, $C \leq \lceil 2. \sum_{i=1} s_i \rceil$.

3. En déduire que First-Fit est à un facteur 2 de l'optimal (2-approximation relative).

Solution: On a $C \leq \lceil 2 \sum_{i=1} s_i \rceil \leq 2. \lceil \sum_{i=1} s_i \rceil \leq 2.C^\star$

4. Donner un exemple de valeurs pour les s_i pour lequel FirstFit donne la solution optimale.

Solution: Si on prend $s_i = 0.5$, on a $C = \lceil \sum_{i=1} s_i \rceil$. Ainsi First-Fit atteint le minorant de la question 1 donc donne bien la solution optimale $C^\star$.

5. Donner un exemple d'ordre de parcours des sacs et de valeurs pour les s_i pour lequel FirstFit atteint le ratio 2 (asymptotiquement).

Solution: Soit l'instance constituée de $n = 2m$ objets avec m pair et pour $i = 1, \ldots, m$, $s_{2i-1} = \frac{1}{2}$ et $s_{2i} = \frac{1}{m}$.
Alors, l'algorithme First-Fit qui parcourt les sacs ajoutés dans l'ordre inverse (le dernier sac ajouté d'abord) peut placer toujours un objet de taille $\frac{1}{m}$ dans un sac contenant déjà un seul objet de taille $\frac{1}{2}$. Il utilise alors $C = m$ sacs. Ce pire cas est atteint aussi par un robot qui parcourt les sacs dans un sens, puis dans l'autre, etc.
Un placement optimal groupe les objets de volume $\frac{1}{2}$ deux par deux. Ce placement a besoin de $\frac{m}{2}$ sacs pour les m objets de taille $\frac{1}{2}$ et de 1 sac supplémentaire pour les m objets de taille $\frac{1}{m}$. Ce placement utilise donc $\frac{m}{2} + 1$ sacs. On a donc $\frac{C}{C^\star} = \frac{2}{1+\frac{2}{m}}$ qui tend vers 2 quand m tend vers l'infini.

6. Une analyse plus fine[1] montre qu'avec FirstFIT $C \leq \left\lceil \frac{17}{10} C^\star \right\rceil = \lceil 1.7 C^\star \rceil$ (ce majorant étant atteint). Un raffinement sur FirstFit, nommée FFD pour *First Fit Decreasing*, consiste à d'abord trier les objets par taille décroissante. Les objets sont alors placés un par un, ceux de plus grande taille d'abord, dans le premier sac qui convient. Ce raffinement de FirstFit (dont l'analyse initiale due à D.S. Johnson a été raffinée par G. Dósa) demande d'allouer seulement $C \leq \frac{11}{9} C^\star + \frac{2}{3}$ (majorant atteint), soit un ratio de performance asymptotique (pour C grand) $\frac{11}{9} \simeq 1.22$. De plus, le ratio absolu est $\frac{C}{C^\star} \leq \frac{4}{3} \simeq 1.33$, et atteint pour quasi toutes les valeurs de C. Ceci montre l'intérêt de trier les objets: sans tri, FirstFit a nécessairement un ratio absolu 1.7 (atteint).
FFD est-il plus compliqué à implémenter que FirstFit? quel est le coût?

Solution: La seule différence avec FirstFit est le tri des élements en entrée qui est en $O(n \log n)$. Sinon, l'allocation des sacs pour FirstFit et FFD est la même. On peut la programmer ainsi.
Implantation naîve: on mémorise les sacs déjà alloués dans un tableau (vector) en mémorisant la place libre disponible dans chaque sac. Lorsqu'on a un nouvel objet, on parcourt le tableau jusqu'à trouver un sac de taille suffisante ou sinon, créer un nouveau sac en fin. Si on a i sacs, l'ajout d'un objet coûte $\mathrm{O}(i)$. D'où $\mathrm{O}(n.C)$ qui est majoré par $\mathrm{O}(n^2)$ (attient si tous les objets sont de taille supérieure à 0.51).
Pour améliorer, on peut stocker les sacs dans un tas (binaire par exemple), le sac de plus grand volume disponible étant en racine. A chaque nouvel objet de taille s_i, on

[1] *First Fit bin packing: A tight analysis.* G. Dósa, J. Sgall. STACS, volume 20 of LIPIcs, page 538-549. 2013.

accède le volume disponible v dans le sac en racine. Si $v \geq s_i$, on supprime la racine du tas, et on rajoute un élément de volume disponible $v - s_i$. Sinon, aucun sac ne peut contenir l'objet : une collection triée par volume libre (par exemple un AVL, un arbre rouge-noir un encore un arbre hachage avec seaux ou mieux dans des seaux avec accès par hachage) on incrémente C de 1 et on ajoute un nouveau sac dans le tas de volume disponible $1 - s_i$. Le tas contient au plus C sacs d'où un coût $O(n \log C)$, majoré par $O(n \log n)$.
Une autre implémentation consiste à utiliser un arbre de recherche en triant les sacs par volume disponible dans un arbre de recherche (par exemple AVL, arbre rouge-noir ou B-tree pour le cache); le sac alloué est alors le sac avec le plus petit volume disponible pouvant contenir l'objet. Le coût de la recherche est alors $O(n \log C)$, comme pour l'implantation de FirstFit par tas et dominé par le coût du tri en entrée des n objets.