M1 Informatique
X1II030

Université de Nantes
2017/2018

Feuille de Travaux Dirigés n^o 3
Approximation et Inapproximation

Exercice 3.1 (Approximation de MINIMUM VERTEX COVER)
On rappelle la définition du problème MIN-VERTEX-COVER (MIN-VC).

MIN-VERTEX-COVER (MIN-VC) *Instance* : un graphe $G = (V, E)$, où $V = \{v_1, v_2 \dots v_n\}$ *Solution* : un ensemble $V' \subseteq V$ qui couvre toutes les arêtes de G *Mesure* : le nombre $\|V'\|$ de sommets dans V'

Voici la description d'un programme linéaire en nombre entiers (qu'on appellera (IP)) :

$$\begin{aligned} \text{minimiser } & x_1 + x_2 + \ldots + x_n & \\ \text{sous les contraintes } & x_i + x_j \geq 1 & \forall (v_i, v_j) \in E(G) \\ & x_i \in \{0, 1\} & \forall v_i \in V(G) \end{aligned}$$

1. Indiquer comment, partant d'une solution de (IP), on peut construire une solution de MIN-VC qui a le même optimum. Justifier.
2. Indiquer comment, partant d'une solution de MIN-VC, on peut construire une solution de (IP) qui a le même optimum. Justifier.

Les questions 1. et 2. ci-dessus montrent donc que les problèmes (IP) et MIN-VC sont équivalents.
Voici maintenant la description d'un programme linéaire (qu'on appellera (LP)). On remarquera qu'ici les y_i ne sont pas nécessairement des entiers, mais peuvent prendre des valeurs quelconques entre 0 et 1.

$$\begin{aligned} \text{minimiser } & y_1 + y_2 + \ldots + y_n & \\ \text{sous les contraintes } & y_i + y_j \geq 1 & \forall (v_i, v_j) \in E(G) \\ & 0 \leq y_i \leq 1 & \forall v_i \in V(G) \end{aligned}$$

Les problèmes de type (LP) peuvent se résoudre en temps polynomial, alors que les problèmes de type (IP) sont NP-complets. L'idée est donc d'utiliser le problème (LP) pour obtenir une approximation du problème (IP), et donc une approximation du problème MIN-VC.

3. Montrer que, pour tout graphe G, $opt(LP) \leq opt(IP)$.

Appelons $y^* = (y_1^*, y_2^* \dots y_n^*)$ une solution optimale obtenue par (LP). On définit alors une solution $x^A = (x_1^A, x_2^A \dots x_n^A)$ pour (IP) de la manière suivante : pour tout $1 \leq i \leq n$, $x_i^A = 1$ si $y_i^* \geq \frac{1}{2}$, et $x_i^A = 0$ sinon.

4. Montrer que si y^* est une solution optimale pour (LP), alors x^A est une solution pour (IP).
5. Montrer que pour tout $1 \leq i \leq n$, $x_i^A \leq 2y_i^*$.
6. Soit $opt(LP) = y_1^* + y_2^* + \dots y_n^*$, et $sol(IP) = x_1^A + x_2^A + \dots x_n^A$. Montrer que $sol(IP) \leq r \cdot opt(LP)$, où r est une valeur à déterminer.
7. Conclure quant à l'approximabilité de (IP) et de MIN-VC.
8. Indiquer (en français) les grandes étapes de l'algorithme d'approximation de ratio r pour MIN-VC que nous venons d'étudier.

Exercice 3.2 (MIN-MAKESPAN)

Soit le problème de minimisation suivant, appelé MIN-MAKESPAN :

> MIN-MAKESPAN
> *Instance* : n tâches de durées $d_1, d_2 \dots d_n$; m machines identiques $M_1, M_2 \dots M_m$, chaque machine ne pouvant réaliser qu'une tâche à la fois
> *Solution* : Une affectation des n tâches aux m machines
> *Mesure* : le temps de terminaison T de l'ensemble des tâches (aussi appelé *makespan*)

Dans la Figure 1, on trouve un exemple de réalisation à $m = 3$ machines (M_1, M_2, M_3), avec $n = 11$ tâches dont les durées sont $d_1 = 2$, $d_2 = 7$, $d_3 = 1$, $d_4 = 3$, $d_5 = 2$, $d_6 = 6$, $d_7 = 2$, $d_8 = 3$, $d_9 = 6$, $d_{10} = 2$, $d_{11} = 5$, et pour lequel $T = 17$ (on ne prétend pas ici que cette réalisation soit optimale).

Il a été démontré que MIN-MAKESPAN est NP-complet. Dans cet exercice, nous voulons montrer que MIN-MAKESPAN est dans APX. Pour cela, on propose l'algorithme suivant (appelé LSA, pour List Scheduling Algorithm) :

(a) prendre les tâches $1, 2 \dots n$ dans l'ordre dans lequel elles sont données ;
(b) pour tout $1 \leq i \leq n$, affecter la tâche i à la première machine disponible (si plusieurs machines sont disponibles, utiliser celle qui a le plus petit indice).

Pour toute instance I du problème MIN-MAKESPAN, on note $T_{opt}(I)$ la solution optimale, et $T_{\text{LSA}}(I)$ le temps obtenu par l'algorithme LSA.

1. Illustrer l'algorithme LSA sur l'exemple de la Figure 1, en présentant le résultat comme dans cette figure, et en indiquant clairement le temps T_{LSA} obtenu.
2. Démontrer que, pour toute instance I, $T_{opt}(I) \geq \frac{\sum_{i=1}^{n} d_i}{m}$.
3. Démontrer que, pour toute instance I, $T_{opt}(I) \geq \max_{i \in [1;n]} d_i$.
4. Démontrer que, pour toute instance I traitée par LSA, au moins une machine fonctionne en continu jusqu'au temps $T_{\text{LSA}}(I)$.

Soit M_i la machine identifiée à la Question 4., et soit j la dernière tâche effectuée par M_i. Appelons aussi $t_{deb}(j)$ le temps auquel la tâche j démarre sur M_i.

5. Indiquer, sur l'exemple de la Question 1., les valeurs de i et j, ainsi que l'endroit où se situent (dans le schéma) M_i et $t_{deb}(j)$.
6. Démontrer que $\frac{\sum_{i \neq j} d_i}{m} \geq t_{deb}(j)$.
7. Sachant que $T_{\text{LSA}}(I) = t_{deb}(j) + d_j$, en déduire une borne supérieure pour $T_{\text{LSA}}(I)$, et conclure que LSA est un algorithme d'approximation dont vous donnerez le ratio r.
8. Quel est le ratio d'approximation de LSA sur l'exemple de la Figure 1 ?

Supposons avoir l'instance suivante I_0 suivante : m machines ; $n = m^2 + 1$ tâches, avec $d_1 = d_2 = \dots = d_{m^2} = 1$ et $d_{m^2+1} = m$.

9. Que vaut $T_{\text{LSA}}(I_0)$? Justifier.
10. Que vaut $T_{opt}(I_0)$? Justifier.
11. Vers quoi tend le ratio $r_0 = \frac{T_{\text{LSA}}(I_0)}{T_{opt}(I_0)}$ lorsque m tend vers l'infini ? Que concluez-vous de ce résultat ?

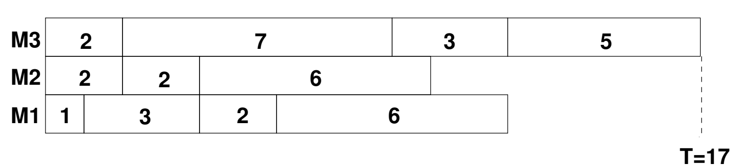

FIGURE 1 – Exemple d'affectation des tâches aux machines, aboutissant à un temps $T = 17$

Exercice 3.3 (MINIMUM BIN-PACKING)

Soit le problème de minimisation suivant, appelé MINIMUM BIN-PACKING (ou MIN-BP) :

> MINIMUM BIN-PACKING (MIN-BP)
> *Instance* : Un ensemble $S = \{s_1, s_2 \dots s_n\}$ où chaque s_i possède un poids $w_i \in \mathbb{N}$, un entier C
> *Solution* : Une partition $P = \{S_1, S_2, \dots, S_m\}$ de S telle que pour tout $1 \leq j \leq m$, $\sum_{s_k \in S_j} w_k \leq C$
> *Mesure* : m, le nombre d'éléments de la partition P

On propose l'algorithme suivant, appelé First-Fit : l'indice j prend la valeur 1. Pour i allant de 1 à n, mettre s_i dans le premier ensemble S_j disponible (c'est-à-dire l'ensemble S_j pouvant encore contenir s_i, et qui est d'indice j le plus petit). Si aucun ensemble S_j ne vérifie ces deux conditions, on en crée alors un nouveau, dans lequel on met s_i.

1. Simuler l'exécution de l'algorithme First-Fit dans le cas où $n = 9$ (donc $S = \{s_1, s_2 \dots s_9\}$), où les poids des s_i sont dans l'ordre : $4; 1; 6; 3; 5; 3; 3; 3; 1$, et où $C = 10$. Donner la valeur de m obtenue, et indiquer ce que contiennent chacun des S_j, $1 \leq j \leq m$.
2. Peut-on faire mieux que l'algorithme First-Fit sur l'exemple de la question précédente ? (si oui, indiquer une meilleure solution ; si non, expliquer pourquoi).
3. Indiquer le ratio d'approximation de l'algorithme First-Fit sur l'exemple de la Question 3.

Soit I une instance quelconque de MIN-BP, et $opt(I)$ le nombre m minimum pour le problème appliqué à I.

4. Donner une borne inférieure "évidente" pour $opt(I)$, et appelons-la $inf(I)$.

Soit $sol_{FF}(I)$ le nombre m obtenu par l'algorithme First-Fit pour le problème MIN-BP appliqué à une instance I donnée.
Soit S_p et S_{p+1}, $1 \leq p < sol_{FF}(I)$, deux ensembles *consécutifs* obtenus par First-Fit sur une instance I donnée. On notera W_p la somme des poids des éléments contenus dans S_p, et W_{p+1} la somme des poids des éléments contenus dans S_{p+1}.

5. Donner une borne inférieure pour la valeur $W_p + W_{p+1}$.
6. Montrer que First-Fit est un algorithme d'approximation de ratio r constant, et indiquer la valeur de r.

Exercice 3.4 (Inapproximabilité de MINIMUM BIN-PACKING)

Soit le problème *de décision* suivant, appelé PARTITION :

> PARTITION
> *Instance* : Un ensemble $S = \{s_1, s_2 \dots s_m\}$ où chaque s_i possède un poids $w_i \in \mathbb{N}$
> *Question* : Existe-t-il une partition de S en S_1 et S_2 telle que $\sum_{s_j \in S_1} w_j = \sum_{s_k \in S_2} w_k$?

On veut montrer, par contradiction et en utilisant le problème PARTITION, que le problème MIN-BP n'est pas approximable sous un ratio $\frac{3}{2}$ (sous des hypothèses raisonnables de complexité, appelées HRC).

1. Indiquer comment transformer toute instance I de PARTITION en une instance I' de MIN-BP, de telle manière que l'optimal pour MIN-BP sur I' soit égal à 2 si et seulement si la réponse à PARTITION sur I est OUI.
2. Supposons que MIN-BP soit approximable avec un ratio $\frac{3}{2}$. Que peut-on alors déduire en ce qui concerne le problème PARTITION ?
3. Sachant que PARTITION est NP-complet, en déduire, sous HRC, que MIN-BP n'est pas approximable sous un ratio $\frac{3}{2}$.

Exercice 3.5 (MAX-IS-4 et MIN-VC-4 sont APX-complets)

Voici la définition de trois problèmes :

MAX-3-SAT-B *Instance* : Une formule booléenne sous FNC Φ construite à partir d'un ensemble $X = \{x_1, x_2 \dots x_n\}$ de variables, et telle que : — chaque clause est de taille 3 dans Φ, et — chaque variable apparaît au plus B fois dans Φ *Solution* : Une affectation `Vrai/Faux` à chaque variable de X. *Mesure* : Le nombre k de clauses satisfaites.

MAX-INDEPENDENT-SET-B (MAX-IS-B) *Instance* : Un graphe $G = (V, E)$ de degré maximum B. *Solution* : Un ensemble stable $V' \subseteq V$ de G. *Mesure* : Le nombre de sommets de V'.

MIN-VERTEX-COVER-B (MIN-VC-B) *Instance* : Un graphe $G = (V, E)$ de degré maximum B. *Solution* : Un vertex cover $V' \subseteq V$ de G. *Mesure* : Le nombre de sommets de V'.

On veut d'abord démontrer que MAX-IS-4 est APX-dur, sachant que MAX-3-SAT-3 est APX-dur.

1. Proposer une réduction de MAX-3-SAT-3 vers MAX-IS-4 qui préserve l'approximation. Plus précisément, pour toute instance I de MAX-3-SAT-3, construire une instance I' de MAX-IS-4, pour laquelle on cherchera à prouver que k clauses seront satisfaites dans $I \Leftrightarrow$ il existe un ensemble indépendant à k sommets dans I'.
2. MAX-IS-4 est-il APX-complet ? Justifier.

Soit G un graphe à n sommets, p la taille de son plus grand ensemble stable (independent set), et q la taille de son plus petit vertex cover.

3. Montrer que $n = q + p$.
4. En déduire que dans tout graphe de degré maximum 4, s'il existe un ensemble stable de taille au moins p, alors il existe un vertex cover de taille au plus $c \cdot p$, où c est une constante à déterminer.
5. En déduire que MIN-VC-4 est APX-dur.
6. MIN-VC-4 est-il APX-complet ? Justifier.