TD n° 12

Algorithmes d'approximation

Hugo Salou MPI*

Dernière mise à jour le 30 janvier 2023

1 Un problème proche de Knapsack

1.

$$\textsc{SommeMax} : \begin{cases} \textbf{Entrée} & : \text{Un entier } n \in \mathbb{N}, \text{ une suite finie } (a_i)_{i \in [\![1,n]\!]} \in \mathbb{N}^n, \\ & \text{ un entier } B \in \mathbb{N} \text{ et un seuil } K \in \mathbb{N}, \\ \textbf{Sortie} & : \text{Existe-t-il } I \subseteq [\![1,n]\!] \text{ avec } K \geqslant \sum_{i \in I} a_i \geqslant B\,? \end{cases}$$

2. On a SommeMax $\preccurlyeq_{\mathrm{p}}$ Knapsack, mais on ne peut pas faire une réduction.
3. Soit $(w_1, \dots, w_n)$ les entrées du problème Partition. On pose $K = B = \sum_{i=1}^n a_i$. On construit l'entrée $(n, (2w_i)_{i \in [\![1,n]\!]}, B, K)$ de SommeMax.

$$\begin{aligned} (n, (2w_i)_{i \in [\![1,n]\!]}, K, B) \in \textsc{SommeMax}^+ &\iff \exists I \subseteq [\![1,n]\!],\ B \leqslant \textstyle\sum_{i \in I} 2a_i \leqslant K \\ &\iff \exists I \subseteq [\![1,n]\!],\ \textstyle\sum_{i=1}^n a_i \leqslant \sum_{i \in I} 2a_i \leqslant \sum_{i=1}^n a_i \\ &\iff \exists I \subseteq [\![1,n]\!],\ \textstyle\sum_{i \in I} 2a_i = \sum_{i=1}^n a_i \\ &\iff \exists I \subseteq [\![1,n]\!],\ \textstyle\sum_{i \in I} a_i = \sum_{i \in [\![1,n]\!] \setminus I} a_i \\ &\iff (w_1, \dots, w_n) \in \textsc{Partition}^+ \end{aligned}$$

Or, comme Partition est **NP**-difficile (*c.f.* TD 8), on en déduit que SommeMax est **NP**-difficile.

4.

Algorithme 1 Algorithme glouton pour résoudre le problème $\textsc{SommeMax}_{\mathrm{O}}$ en $\mathcal{O}(n)$

```
somme ← 0
pour i ∈ ⟦1, n⟧ faire
    si somme + a_i ⩽ B alors
        somme ← somme + a_i
retourner somme
```

5. L'algorithme n'est pas optimal : avec l'entrée $(1, B)$, l'algorithme renvoie 1, mais la valeur optimale est B. Cet algorithme n'est pas une ρ-approximation, pour $\rho \in \mathbb{R}$.
6.

Algorithme 2 Algorithme glouton pour résoudre le problème $\textsc{SommeMax}_{\mathrm{O}}$ en $\mathcal{O}(n \ln n)$

```
On trie, par ordre décroissant, les entrées (a_1, ..., a_n) avec un tri rapide.
somme ← 0
pour i ∈ ⟦1, n⟧ faire
    si somme + a_i ⩽ B alors
        somme ← somme + a_i
retourner somme
```

7. Soit e une entrée. Soit S la solution, *non nécessairement optimale*, de l'algorithme. Soit $S^\star$ la solution optimale. On a $S^\star \leqslant B$.
   - Si $S \geqslant \frac{B}{2}$, alors $\frac{S}{S^\star} \geqslant \frac{B/2}{B} = \frac{1}{2}$.
   - Si $S < \frac{B}{2}$, alors, en supposant que $(a_1, \dots, a_n)$ est trié par ordre décroissant, on a, pour $i \in [\![1,n]\!]$, $a_i > B$ ou $a_i < \frac{B}{2}$.
     - Si $\forall i \in [\![1,n]\!]$, $a_i > B$, alors $S = 0 = S^\star$.
     - Soit $i = \arg\min_{i \in [\![1,n]\!]}(a_i < B/2)$. On pose I_{algo} les valeurs de i telles que a_i ait été ajoutée à somme. On pose $i' = \max I_{\mathrm{algo}}$.

       Si $i' < n$, alors par l'algorithme, l'objet d'indice n ne rentre pas dans le sac. Or, la valeur du sac est inférieure stricte à $B/2$, et $a_n < B/2$, ce qui est absurde (l'objet rentre dans le sac).

       Ainsi, $i' = n$, et l'algorithme a mis dans le sac tous les objets de poids inférieurs ou égal à $B/2$, donc tous les objets ont un poids inférieur ou égal à B. Donc $S^\star = S$.

8.

Algorithme 3 Algorithme glouton pour résoudre le problème SommeMax$_O$ en $\mathcal{O}(n)$

Entrée $(a_i)_{i\in[\![1,n]\!]}$ une suite finie

somme $\leftarrow 0$
maxi $\leftarrow 0$
pour $i \in [\![1,n]\!]$ **faire**
  si $a_i \geqslant B/2$ et $a_i \leqslant B$ **alors**
    maxi $\leftarrow a_i$
  sinon si $a_i + \text{somme} \leqslant B$ **alors**
    somme $\leftarrow$ somme $+ a_i$
retourner $\max(\text{somme}, a_i)$

2 Le problème BinPacking

1.

$$\text{BinPacking}_O : \begin{cases} \textbf{Entrée} & : \text{un entier } n \in \mathbb{N}\text{, une suite finie } (t_i)_{i\in[\![1,n]\!]} \in \mathbb{N}^n \text{ et } C \in \mathbb{N}^* \\ \textbf{Sortie} & : \max\Big\{K \in \mathbb{N} \mid \exists\varphi : [\![1,n]\!] \to [\![1,K]\!],\ \forall i \in [\![1,K]\!],\ \sum_{j\in\varphi^{-1}(\{i\})} t_k \leqslant C\Big\} \end{cases}$$

2. On réduit Partition à BinPacking. On construit les entrées à l'aide de l'algorithme ci-dessous.

Algorithme 4 Réduction polynômiale de Partition à BinPacking

$B \leftarrow \sum_{i=1}^{n} t_i$
si $B \equiv 0\ [2]$ et $B \neq 0$ **alors**
  retourner $(n, T, B/2, 2)$
sinon
  retourner $(3, (1,1,1), 2, 1)$

Montrons que BinPacking $\in$ **NP**. On choisit pour ensemble de certificats l'ensemble des fonctions $\varphi : [\![1,n]\!] \to [\![1,K]\!]$, pour $(n,K) \in \mathbb{N}^2$. On devine un certificat φ, on vérifie que $\forall i \in [\![1,K]\!]$, $\sum_{j\in\varphi^{-1}(\{i\})} t_j \leqslant C$ en temps polynômiale.

3.

4. On considère l'entrée $(3,(3,3,3),3)$ de l'algorithme. La solution optimale est 3, et c'est la réponse donnée par l'algorithme. On considère l'entrée $(3,(2,3,1),3)$ de l'algorithme. La solution optimale est 2, mais l'algorithme renvoie 3.

5. Pour une instance (n,t,C), un minorant est $\left\lceil \frac{V}{C} \right\rceil$ avec $V = \sum_{i=1}^{n} t_i$, un majorant est n.

3 Le problème VoyageurCommerce

1. Soit $\mathscr{C} = \{a_1,a_2\} - \cdots - \{a_n,a_{n+1}\}$ et $a_{n+1} = a_1$, un tour où $\bigcup_{i\in[\![1,n]\!]}\{a_i\} = S$. Soit $\{a,b\}$ une arrête. Posons $\mathscr{C}' = \mathscr{C} \setminus \{\{a,b\}\}$, et $H = (S', \mathscr{C}')$, où $S' = S$. Montrons que H est un arbre couvrant.

   — Montrons que H est connexe. Soit un couple sommets $(x,y) \in S^2$ tel que $x \neq y$. $\mathscr{C}$ est un tour, donc en re-numérotant

   $$x - b_1 - b_2 - \cdots - b_{n-1} - x.$$

   Soit $i \in [\![1,n-1]\!]$ tel que $b_i = y$. Soit $j \in [\![1,n]\!]$ tel que $\{b_j, b_{j+1}\} = \{a,b\}$.

   Cas 1 Si $j < i$, alors $y - b_{i+1} - \cdots - b_{n-1} - x$ est un chemin.

   Cas 2 Si $j > i$, alors $x - b_1 - \cdots - b_{i-1} - y$ est un chemin.

   Donc H est connexe.

   — On a $\#\mathscr{C}' = n-1$ et H est connexe, donc acyclique.

— Enfin $S' = S$, donc H est couvrant.

2. Soit $T^\star$ le tour de coût minimal. On note $c(T)$ le coût d'un tour T. On a $c(T^\star) \geqslant c(H)$ où H est l'arbre défini dans la question précédente (par suppression d'arête), d'où $c(T^\star) = c(H^\star)$ où $H^\star$ est un arbre couvrant de poids minimal (par définition de poids minimal).
3. Non, avec le graphe ci-dessous est un contre-exemple.

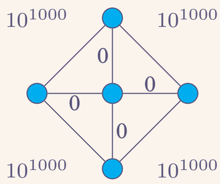

Figure 1 – Contre exemple

4. On obtient l'arbre couvrant de poids minimal suivant.

Figure 2 – Arbre couvrant de poids minimal

Un parcours en profondeur est

$$a \to e \to c \to d \to h \rightsquigarrow g \to b \to f.$$