Synthèse du cours 7 : programmation dynamique (4)

25 octobre 2022

François Laroussinie

NB : Ces synthèses ont pour but de compléter les notes prises en cours. Elles ne les remplacent pas ! En particulier, la plupart des preuves n'y figurent pas. Rappel : il faut programmer les algorithmes vus en cours.

1 Construction d'un ABR optimal

On rappelle que dans un ABR (arbre binaire de recherche), on stocke un élément dans chaque noeud de l'arbre et que l'on assure que si x' est dans le sous-arbre gauche de x alors $x' \leq x$, et si x' est dans le sous-arbre droit de x alors $x' > x$.

L'idée est de construire un ABR optimal à partir d'un ensemble d'éléments dont on connait la fréquence (de recherche). Par optimal, on entend un ABR A tel que le coût total de la recherche des n éléments selon leur fréquence respective soit minimal, c'est-à-dire que la somme $\sum_{x:\text{élt}}((\mathsf{prof}_A(x)+1)\cdot \mathsf{freq}(x))$ soit minimale (où $\mathsf{prof}_A(x)$ désigne la profondeur du noeud de A correspondant à l'élément x et $\mathsf{freq}(x)$ la fréquence de x). Notons que $(\mathsf{prof}(x)+1)$ correspond au coût (en nombre de comparaisons) de la recherche de l'élément x dans l'ABR. Formellement, on a donc :

Input : n valeurs $x_1, \cdots, x_n$ ordonnées ($x_1 \leq \cdots \leq x_n$) et n fréquences $f_1, \cdots, f_n$.
Output : Un ABR contenant les valeurs x_i minimisant la somme $\sum_{x:\text{élt}}((\mathsf{prof}(x)+1)\cdot \mathsf{freq}(x))$.

Pour résoudre ce problème, on va calculer le coût minimal $d_{i,j}$ (avec $i \leq j$) d'un ABR contenant les éléments $x_i, x_{i+1}, \ldots, x_j$.

La valeur de $d_{i,j}$ va dépendre du choix de l'élément positionné à la racine de l'ABR. Supposons que ce soit l'élément x_k avec $i \leq k \leq j$, alors le sous-arbre gauche (un ABR optimal !) contiendra les éléments $x_i, \ldots, x_{k-1}$, et le sous-arbre droit contiendra les éléments $x_{k+1}, \ldots, x_j$.

On peut montrer (cf le cours) que :

$$d_{i,j} = \sum_{i \leq e \leq j} \mathsf{freq}(x_e) + \min_{i \leq k \leq j}(d_{i,k-1} + d_{k+1,j})$$

avec comme convention $d_{i,i} = \mathsf{freq}(x_i)$ et $d_{i,j} = 0$ si $i > j$. Ensuite il suffit d'énumérer correctement les indices... Voir l'algorithme 1.

Pour obtenir l'ABR optimal, il suffit de stocker la racine choisie pour chaque paire (i,j)...

2 Un algorithme d'approximation pour le problème du sac à dos

Pour cette partie, on peut consulter le livre *Algorithms* de Dasgupta, Vazirani et Papadimitriou.

Procédure `ABRopt` (X, F)
begin
  `D` : matrice d'entiers de taille $|X| \times |X|$, initialisée avec ∞
  pour $i = 0 \dots |X|$ **faire** $\texttt{D}[i, i] = F[i]$
  pour $d = 1 \dots |X| - 1$ **faire**
    pour $p = 0 \dots |X| - d - 1$ **faire**
      $i := p, \quad j := d + p$
      pour $k = i \dots j$ **faire**
        `aux` $:= 0$
        si $i \leq k - 1$ **alors** $\texttt{aux} += \texttt{D}[i][k-1]$
        si $k + 1 \leq j$ **alors** $\texttt{aux} += \texttt{D}[k+1][j]$
        $\texttt{D}[i][j] := \min(\texttt{D}[i][j], aux)$
      $\texttt{D}[i][j] := \texttt{D}[i][j] + \sum_{i \leq u \leq j} F[u]$
  return $\texttt{D}[0, |X| - 1]$

Algorithme 1 : algorithme pour calculer le coût d'ABR optimal

2.1 Des algorithmes d'approximation

Ici nous allons voir un « schéma d'approximation *entièrement* polynomial » pour le problème du sac à dos. Pour les problèmes d'optimisation, on part d'une instance du problème à résoudre (ici le sac à dos) et d'un paramètre $\varepsilon > 0$. L'algorithme va calculer un résultat s dont on garantit qu'il est supérieur ou égal à $(1 - \varepsilon) \cdot s_{opt}$ pour un problème de maximisation (et inférieur ou égal à $(1 + \varepsilon) \cdot s_{opt}$ pour un problème de minimisation) où s_{opt} est la solution optimale, et de plus ce calcul va se faire en temps polynomial en la taille de l'instance du problème et polynomial dans $\frac{1}{\varepsilon}$.

2.2 Application au sac à dos

On considère la variante sans répétition.

Input : Un poids maximal $W \in \mathbb{N}$, un ensemble de n objets ayant chacun un poids w_i et une valeur v_i pour $i = 1, \dots, n$. On suppose $w_i > 0$ et $v_i > 0$.
Output : La **valeur** maximale pouvant être stockée dans le sac sans dépasser sa capacité W.

Dans la version classique, on calcule le tableau $K[i, w]$, avec $0 \leq i \leq n$ et $0 \leq w \leq W$, défini comme étant la **valeur** maximale pouvant être stockée dans un sac de **capacité** w avec les objets $1, 2, 3, \dots, i$. Le résultat du problème est alors la valeur $K[n, W]$: la valeur maximale pour un sac de capacité W en prenant des objets parmi tous les objets possibles. La complexité de l'algorithme est alors en $O(n.W)$.

Sur le même modèle, on peut concevoir un algorithme en $O(V \cdot n)$, où $V = v_1 + v_2 + \ldots + v_n$, en construisant une table $\mathsf{T}[\mathsf{i}, \mathsf{v}]$ stockant le **poids minimal** pour réaliser la valeur v en choisissant des objets parmi le sous-ensemble $\{1, \dots, i\}$. On a ainsi pour $i, v > 0$:

$$\mathsf{T}[i, v] = \min(\mathsf{T}[i-1, v], w_i + \mathsf{T}[i-1, v - v_i])$$

avec $\mathsf{T}[i, 0] = 0$ et $\mathsf{T}[0, v] = +\infty$ pour $v > 0$. Ensuite il reste à trouver le plus grand v tel que $\mathsf{T}[n, v] \leq W$ (donc en parcourant la dernière ligne de la table). Enfin, on peut facilement étendre cette procédure pour renvoyer le sous-ensemble d'objets pour obtenir cette valeur

$\mathsf{T}[n, v]$. C'est cette procédure que l'on utilise dans l'algorithme d'approximation décrit ci-dessous.

Procédure `ApproxSaD` (E, W, ε)
begin
Retirer de E les objets de poids supérieur strictement à W
$n = |E|$
$v_M := \max_i(v_i)$
pour $i = 1 \dots n$ **faire** $\widehat{v_i} := \lfloor v_i \cdot \frac{n}{\varepsilon \cdot v_M} \rfloor$
Soit $\widehat{E} = \{(\bar{v_1}, w_1), \dots, (\bar{v_n}, w_n)\}$
Renvoyer le résultat de l'algorithme en $O(n.\widehat{V})$ sur $\widehat{E}$
Algorithme 2 : algorithme d'approximation pour le problème du sac à dos.

Comme on a $\widehat{v_i} \leq \frac{n}{\varepsilon}$ (car $\frac{v_i}{v_M} \leq 1$) pour tout i, on a $\widehat{V} = \sum_i \widehat{v_i} \leq \frac{n^2}{\varepsilon}$ et la complexité de l'algorithme ApproxSaD est clairement en $O(\frac{n^3}{\varepsilon})$. Donc c'est bien un algorithme avec la complexité souhaitée.

Il reste à montrer sa correction, c'est-à-dire que le résultat renvoyé est supérieur à $K_{opt}(1-\varepsilon)$ où K_{opt} est la solution optimale pour l'instance de départ (E, W). Soit $S_{opt} \subseteq \{1, \dots, n\}$ un sous-ensemble d'objets qui réalise la valeur K_{opt}.

Que vaut $\widehat{v}(S_{opt})$, c'est-à-dire la valeur de S_{opt} selon les valeurs de $\widehat{E}$?

$$\begin{aligned} \widehat{v}(S_{opt}) = \sum_{i \in S_{opt}} \widehat{v_i} \;&=\; \sum_{i \in S_{opt}} \lfloor v_i \cdot \frac{n}{\varepsilon \cdot v_M} \rfloor \;\geq\; \sum_{i \in S_{opt}} (v_i \cdot \frac{n}{\varepsilon \cdot v_M} - 1) \\ &\geq\; (\frac{n}{\varepsilon \cdot v_M} \sum_{i \in S_{opt}} v_i) - |S_{opt}| \;\geq\; (\frac{n}{\varepsilon \cdot v_M} \sum_{i \in S_{opt}} v_i) - n \;=\; \frac{n}{\varepsilon \cdot v_M} \cdot K_{opt} - n \end{aligned} \tag{1}$$

Considérons maintenant l'ensemble S_{algo} renvoyé par l'algorithme qui est optimal pour $\widehat{E}$ et donc pour lequel nous avons :

$$\sum_{i \in S_{algo}} \widehat{v_i} \;\geq\; \sum_{i \in S_{opt}} \widehat{v_i} \;\geq\; \frac{n}{\varepsilon \cdot v_M} \cdot K_{opt} - n$$

Comme $\widehat{v_i} = \lfloor v_i \cdot \frac{n}{\varepsilon \cdot v_M} \rfloor$, on a $\widehat{v_i} \leq \frac{n \cdot v_i}{\varepsilon \cdot v_M}$ et donc $v_i \geq \frac{\widehat{v_i} \cdot \varepsilon \cdot v_M}{n}$. On peut alors estimer la valeur de S_{algo} selon les valeurs de E (c'est-à-dire le résultat renvoyé par : ApproxSaD) :

$$\begin{aligned} \sum_{i \in S_{algo}} v_i \;&\geq\; \sum_{i \in S_{algo}} \frac{\widehat{v_i} \cdot \varepsilon \cdot v_M}{n} = \frac{\varepsilon \cdot v_M}{n} \sum_{i \in S_{algo}} \widehat{v_i} \;\geq\; \frac{\varepsilon \cdot v_M}{n} \big(\frac{n}{\varepsilon \cdot v_M} \cdot K_{opt} - n \big) \\ &\geq\; K_{opt} - \varepsilon \cdot v_M \;\geq\; K_{opt} - \varepsilon \cdot K_{opt} \;=\; K_{opt} \cdot (1 - \varepsilon) \end{aligned} \tag{2}$$

Notons que l'on utilise bien le fait que les objets de poids supérieurs à W ont été retirés de E pour déduire $v_M \leq K_{opt}$.

Exemple : Si on prend l'exemple suivant (on ne donne pas les poids) et $\varepsilon = 0.01$ (on cherche donc une solution à 1% près, on obtient les valeurs $\widehat{v_i}$ ci-dessous. Cela illustre la simplification faite par l'algorithme...

v_1	v_2	v_3	v_4
130425108	250133075	75205010	50405013

$\widehat{v_1}$	$\widehat{v_2}$	$\widehat{v_3}$	$\widehat{v_4}$
208	400	120	80