Synthèse du cours 6 : programmation dynamique (3)
et algorithme d'approximation

5 novembre 2024

François Laroussinie

1 Un jeu

On considère un jeu à deux joueurs. Au début du jeu, on dispose une suite de n cartes $c_1,\ldots, c_n$ sur une table. Chaque carte c_i a une valeur $v_i \in \mathbf{R}$. A chaque tour, le joueur 1 choisit une des deux cartes situées à une extrémité (gauche ou droite) de la rangée. Le joueur 2 fait ensuite de même. Le jeu s'arrête lorsque toutes les cartes ont été prises par les deux joueurs. Le gagnant est celui dont la somme de ses cartes est la plus grande.

A chaque tour, la position du jeu est définie par une paire (une *configuration*) (i, j) où $1 \leq i \leq j \leq n$: la carte la plus à gauche est c_i et celle la plus à droite est c_j. La configuration initiale est $(1, n)$. L'objectif est de calculer la meilleure stratégie possible pour le joueur 1.

On peut construire une table $\mathsf{TV}[-,-]$ de dimension $n \times n$ telle que pour toute configuration (i, j) $\mathsf{TV}[i, j]$ contient la valeur optimale que peut **assurer** le joueur 1 lorsqu'il a le trait (NB : « assurer » signifie ici que le joueur 1 peut obtenir cette valeur (ou plus) quoi que fasse le joueur 2) :

$$\mathsf{TV}[i,j] \;=\; \begin{cases} 0 & \text{si } i > j \\ v_i & \text{si j=i} \\ \max(v_i, v_j) & \text{si j=i+1} \\ \max(v_i + \min(\mathsf{TV}[i+2,j], \mathsf{TV}[i+1,j-1]), & \\ \qquad v_j + \min(\mathsf{TV}[i,j-2], \mathsf{TV}[i+1,j-1])) & \text{sinon} \end{cases}$$

Il reste à énumérer correctement des paires i, j afin de remplir le tableau $\mathsf{TV}[i, j]\ldots$ A faire en exercice !

Une fois calculé $\mathsf{TV}[1, n]$, on peut répondre à la question « est-ce que le joueur 1 dispose d'une stratégie gagnante ou non ? » en comparant $\mathsf{TV}[1, n]$ et $(\sum_{i=1..n} c_i) - \mathsf{TV}[1, n]$.

2 Algorithmes d'approximation

Schéma d'approximation polynomial. Pour les problèmes d'optimisation, on appelle un schéma d'approximation entièrement polynomial (abréviation anglaise : EPTAS) un algorithme qui prend en entrée une instance du problème à résoudre (de taille n) et un paramètre $\varepsilon > 0$ et calcule 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 ce calcul se fait en temps polynomial en la taille n et polynomial en $\frac{1}{\varepsilon}$.

Application au sac à dos. On trouvera cet algorithme expliqué dans *Algorithms* de Dasgupta, Vazirani et Papadimitriou. On note V la somme des valeurs $v_1+\ldots+v_n$. Précédemment on a vu un algorithme en $O(W \cdot n)$ pour résoudre le problème. Sur le même modèle, on peut concevoir un algorithme en $O(V \cdot n)$ 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 $\{1,\ldots,i\}$. 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).

A partir de cet algorithme, on peut définir l'algorithme d'approximation `ApproxSaD` qui prend en argument une instance (E, W) du sac à dos et un paramètre ε :

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 $i = 1 \ldots n$ **faire** $\widehat{v_i} := \lfloor v_i \cdot \frac{n}{\varepsilon \cdot v_M} \rfloor$
Soit $\widehat{E} = \{(\bar{v_1}, w_1), \ldots, (\bar{v_n}, w_n)\}$
Renvoyer le résultat de l'algorithme en $O(n.\widehat{V})$ sur $\widehat{E}$

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 polynomial.

Il reste à montrer que le résultat est supérieur à $K(1-\varepsilon)$ où K est la solution optimale pour l'instance de départ (E, W). Soit $S_{opt} \subseteq \{1,\ldots,n\}$ 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 :

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

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

Approximation de subsetsum. Cet algorithme est décrit dans « Introduction à l'algorithmique » de Cormen, Leiserson, Rivest et Stein. On peut le décrire comme suit où $\mathsf{Fusion}(L, L')$ consiste à fusionner deux listes triées. Notons aussi que dans la mesure où les listes L_i sont triées, la dernière instruction revient à renvoyer le dernier élément de L_n.

Procédure `Approx-SubsetSum` (E, V, ε)
begin
  $n = |E|$
  $L_0 = [0]$
  pour $i := 0 \dots n-1$ **faire**
    $L_i = \mathsf{Fusion}(L_{i-1}, x_i + L_{i-1})$
    $L_i = \mathsf{Reduction}(L_i, \frac{\varepsilon}{2 \cdot n})$
  return $\max(L_n)$

Procédure `Reduction` (L, s)
begin
  $m = |L|$
  $res = [L[0]]$
  $last = res$
  pour $i := 1 \dots n-1$ **faire**
    si $L[i] > last \cdot (1+s)$ **alors**
      $res.\mathsf{append}(L[i])$
  return res

Il reste à montrer que l'algorithme `Approx-SubsetSum` donne une solution acceptable (*i.e.* entre $(1-\varepsilon)V_{opt}$ et V_{opt}) et procède en temps polynomial dans la taille du problème (*i.e.* n et $\log V$) et $\frac{1}{\varepsilon}$. Dans la suite on notera L_i les listes calculées par l'algorithme optimal qui suit la même idée sans appliquer de réduction, et L'_i celles de l'algorithme d'approximation `Approx-SubsetSum`.

Pour le voir, on peut montrer par induction sur i, que pour tout élément y de L_i, il existe un $z \in L'_i$ tel que $\frac{y}{(1+\frac{\varepsilon}{2n})^i} \leq z \leq y$. On enlève donc des valeurs, mais pour chaque valeur manquante, il reste dans L'_i une valeur assez proche de celle supprimée.

En particulier, c'est vrai pour les valeurs renvoyées V_{opt} et V_{algo}, on a : $\frac{V_{opt}}{(1+\frac{\varepsilon}{2n})^i} \leq V_{algo} \leq V_{opt}$. On peut montrer $(1+\frac{\varepsilon}{2n})^n \leq 1+\varepsilon$ et en déduire :

$$V_{opt} \leq (1+\varepsilon) \cdot V_{algo}$$

d'où $V_{algo} \geq (1-\varepsilon)V_{opt}$.

Pour la complexité, on peut observer que le nombre de valeurs possibles (donc la taille) d'une liste L_i est bornée par $2 + \lfloor \log_{1+\frac{\varepsilon}{2n}} V \rfloor$ et de plus on peut montrer :

$$\log_{1+\frac{\varepsilon}{2n}} V \ \leq \ \frac{4 \cdot n \cdot \log V}{\varepsilon} + 2$$

Et donc on a un algorithme en $O(\frac{n^2 \cdot \log V}{\varepsilon})$.