HMMA237

Programmation dynamique

Cours: Joseph Salmon *Scribes: Julie Roste et Abdillahi Omar Djama*

Ce cours est inspiré du polycopié rédigé par G. Carlier [Car07] ainsi que de la page Wikipedia https://en.wikipedia.org/wiki/Dynamic_programming.

1 Introduction

Un graphe est composé de nœuds et d'arêtes reliant certains de ces nœuds. On dispose soit des graphes orientés ou des graphes non-orientés. Un graphe orienté est déterminé par :

- un ensemble de nœuds
- un ensemble ordonné de couples de nœuds appelés arêtes.
- les arêtes sont orientées : l'arrête $u = (a, b)$ va de a vers b.

et le non-oriente qui est déterminé par ;

- d'un ensemble de nœuds X.
- d'un ensemble E de paires de nœuds appelées arêtes.
- les arêtes ne sont pas orientées.

2 Exemple canonique : *Le plus court chemin dans un graphe*

Le graphe dispose plusieurs types de représentation dont l'un est la matrice d'adjacence. La matrice d'adjacence d'un graphe comme en Figure 1 est égale à la matrice $A = (a_{i,j})$ de dimension $n \times n$ telle que

$$A = \begin{bmatrix} 0 & 2 & 3 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \quad A_{i,j} = \begin{cases} a_{ij}, & \text{si } i \text{ est relié à } j, \\ 0, & \text{si } i \text{ n'est pas relié à } j. \end{cases} \tag{1}$$

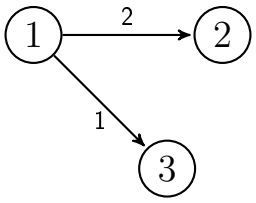


FIGURE 1 – Un graphe de 3 nœuds et 2 arêtes pondérées.

En théorie des graphes, un problème classique est de trouver le chemin le pour aller d'un nœud à un autre de façon que la somme des poids des arêtes (*e.g.*, les distances) de ce chemin soit minimale. Sa résolution illustre de manière simple le principe de la programmation dynamique.

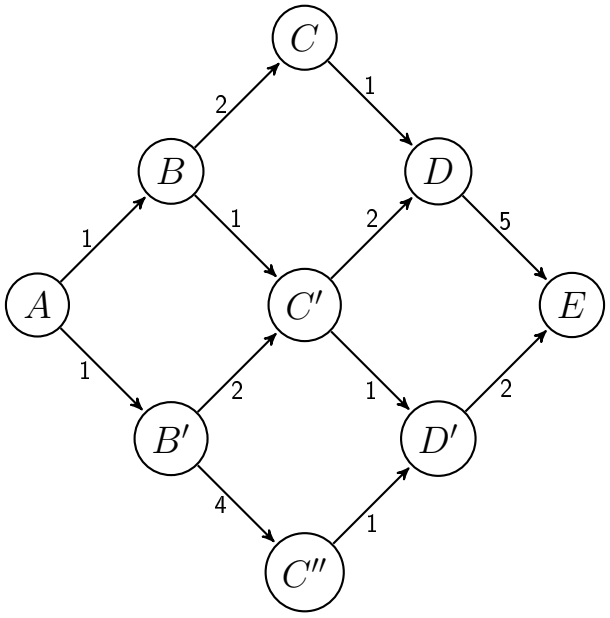


FIGURE 2 – Graphe de A à E avec des arêtes pondérées.

Considérerons un voyageur qui doit se rendre de la ville A à la ville E en passant par plusieurs villes intermédiaires. Les chemins possibles sont donc modélisés par un graphe ayant A et E pour nœuds initial et terminal (les autres nœuds représentant les villes étapes). Les arêtes de ce graphe représentent les trajets intermédiaires, voir Figure 2.

Problématique : Quel est le plus court chemin pour passer de A a E ?
Solution : D'après le graphe le chemin le plus court est $A \to B \to C' \to D' \to E$.

Remarque. *L'approche par force brute est impossible si le nombre d'arêtes augmente trop vite (problème combinatoire).*

Remarque. *Si un chemin est optimal pour aller de A à E, pour tout nœud (ville étape) M de ce chemin, restreint de $A \xrightarrow{C_1} M$ et $M \xrightarrow{C_2} E$ les chemins restreint sont optimaux pour relier A à M et M à E. Enfin $C_1 + C_2$ représente le chemin total [Bel54].*

Démonstration. Par l'absurde : s'il existait un chemin (sous chemin) plus rapide, par exemple de $A \xrightarrow{C_1'} M$, la somme $C_1' + C_2$ est meilleur que la somme $C_1 + C_2$ ce qui est absurde par optimalité du chemin : $C_1 + C_2$. □

3 Rétroaction arrière (En : *Backward induction*)

Introduisons la fonction valeur $V(M) :=$ *temps de parcours minimal entre M et E*. La fonction V se calcule facilement en partant de la fin puis en procédant par rétroaction arrière (english : *backward induction*). On a d'abord :

$$V(D) = 5 \tag{2}$$
$$V(D') = 2 \tag{3}$$
$$\tag{4}$$

On calcule ensuite les villes précédentes qui donne

$$\begin{aligned} V(C) &= 6 \\ V(C') &= \min(2 + V(D), 1 + V(D')) = 3 \\ V(C'') &= 3 \end{aligned}$$

Réitérant l'argument, il vient :

$$\begin{aligned} V(B) &= \min(2 + V(C), 1 + V(C')) = 4 \\ V(B') &= \min(2 + V(C'), 4 + V(C'')) = 5, \end{aligned}$$

et enfin,

$$V(A) = \min(1 + V(B), 1 + V(B')) = 5,$$

d'où la distance minimale de A à E est 5 et correspond au parcours $A \to B \to C' \to D' \to E$.

1. L'algorithme marche pour beaucoup de graphes. (il faut seulement éviter les graphes qui contiennent des cycles à valeurs négatives : on peut alors tendre vers $-\infty$ en passant sur eux une infinité de fois).
2. Si on s'est trompé à une étape (*e.g.*, passer par B' dans le cas de notre exemple), par exemple à cause d'un aléa, on sait encore comment continuer pour résoudre le nouveau problème de chemin optimal. Cet algorithme est dit **robuste** aux erreurs.
3. En pratique on stock le chemin et la valeur optimal.

3.1 Exemple Fibonacci

La suite de Fibonacci est définie par

$$\begin{aligned} F_0 &= 1 \\ F_1 &= 1 \\ F_n &= F_{n-1} + F_{n-2}, \end{aligned} \tag{5}$$

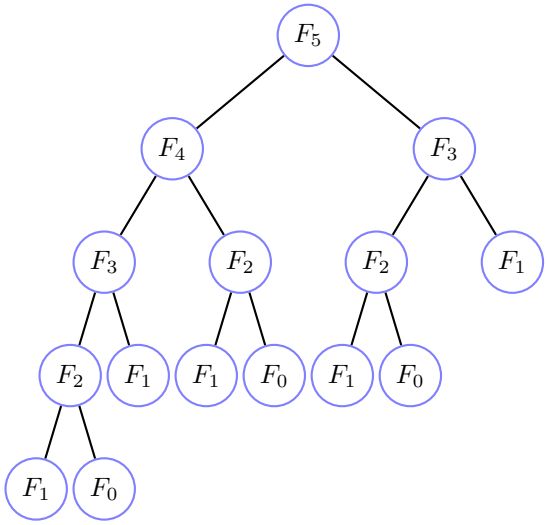
où les 10 premiers nombres de Fibonacci sont $\{1, 1, 2, 3, 5, 8, 13, 21, 34, 55\}$. Pour calculer la suite on peut utiliser un algorithme d'approche brute (voir Figure 3a) ou un algorithme qui stocke les résultats et les réutilise (Figure 3b) où ce dernier est plus efficace.

4 Formulation Mathématique

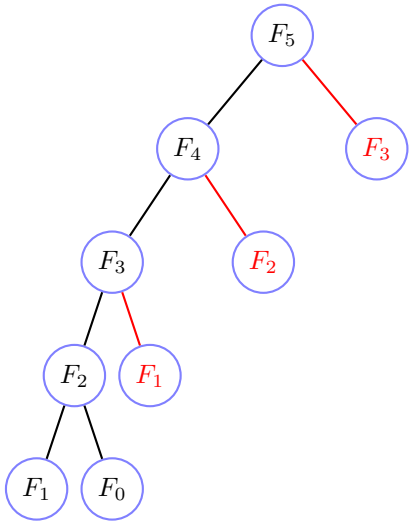
Fixons x et $T \in \mathbb{N}$. On se propose d'étudier les problèmes de programmation dynamique suivant

$$\min_{x_t} \sum_{t=0}^{T-1} V_t(x_t, x_{t+1}) \quad \text{tel que } x_0 = x. \tag{6}$$

V_t correspond au coût entre x_t et x_{t+1}, on peut ajouter avec ce coût le caractère admissible d'un x_{t+1} depuis x_t. Si x_t ne peut pas "engendrer" x_{t+1}, alors $V_t(x_t, x_{t+1}) = +\infty$.



(a) Un graphe pour 5 itérations de Fibonacci, utilisant un algorithme sans memoire.



(b) Un graphe pour 5 itérations de Fibonacci, utilisant un algorithme avec mémoire. Les nœuds rouges sont déjà stocké, et on n'a pas besoin de les calculer à nouveau.

FIGURE 3 – Graphes pour coder la fonction de Fibonacci

Definition. *Compte tenu de la structure récursive du problème il est judicieux d'introduire les fonctions-valeur aux différentes dates. Pour x fixe on définit donc*

$$V(0,x) = \min_{x_t} \left\{ \sum_{t=0}^{T-1} V_t(x_t, x_{t+1}), \quad x_0 = x \right\}$$
$$V(1,x) = \min_{x_t} \left\{ \sum_{t=1}^{T-1} V_t(x_t, x_{t+1}), \quad x_1 = x \right\}$$
$$\vdots$$
$$V(t-1,x) = \min_{x_t} \{ V_{T-1}(x_t, x_{t+1}), \quad x_{T-1} = x \} \quad .$$

Proposition. *Si $(x_0^*, x_1^*, \ldots, x_T^*)$ est optimale pour $V(0,x)$ avec $x_0^* = x$, alors pour tout $\zeta \in [1; T-1]$, la suite $(x_\zeta^*, \ldots, x_T^*)$ est optimale pour $v(\zeta, x_\zeta^*)$.*

Démonstration. v(0,x)=$\sum_{t=0}^{T-1} V_t(x_T^*, x_{T+1}^*)$ avec $(x_0^* = x)$, par absurde s'il existe $\zeta \in [1; T-1]$ telle que $(x_\zeta^*, \ldots, x_T^*)$ n'est pas optimale pour $v(\zeta, x_\zeta^*) = \sum_{t=\zeta}^{T-1} V_t(x_T^*, x_{T+1}^*) > \sum_{t=\zeta}^{T-1} V_t(Z_T, Z_{T+1})$ avec $Z_\zeta = x_\zeta^*$. Prenons (y_T) où $(y_0, \ldots, y_T) = (x_0^*, \ldots, x_{\zeta-1}^*, Z_\zeta, \ldots, Z_T)$ alors

$$\sum_{t=0}^{T-1} V_t(y_T, y_{T+1}) < (\sum_{t=0}^{T-1} V_t(x_T^*, x_{T+1}^*) + \sum_{t=\zeta}^{T-1} V_t(x_T^*, x_{T+1}^*)) = V(0,x) \tag{7}$$

□

Proposition. *Pour tout* $t \in [\![1; T-1]\!]$, $v(t,x) = \min\limits_y \{V_t(x,y) + v(t+1,y)\}$.

Démonstration. Par récurrence en $t = 0$, montre que $V(0,x) = \min\limits_y \{V_t(x,y) + v(1,y)\}$.

($\Rightarrow$) : Fixons (y_t).

$$\begin{aligned} V(0,x) &\leq V_0(x,y) + \sum_{t=1}^{T-1} V_t(y_T, y_{T+1}) \quad \text{avec } y_T = y \\ V(0,x) &\leq V_0(x,y) + v(1,y), \quad \text{choisir } (y_1, \dots, y_T \text{ optimaux}) \\ V(0,x) &\leq \min_y \{V_0(x,y) + v(1,y)\} \end{aligned}$$

Montrons maintenant l'autre inclusion ($\Leftarrow$) :

Prenons $\epsilon > 0$ et $(x_0, \dots, x_{T-1})$, ϵ-optimaux.

$$V(0,x) + \epsilon \geq \sum_{t=0}^{T-1} V_t(x_T, x_{T+1}) = V_0(x_0, x_1) + \sum_{t=1}^{T-1} V_t(x_T, x_{T+1}) \tag{8}$$

Garde en tête que

$$\begin{aligned} &\min_y \{V_0(x,y) + v(1,y)\} \leq V_0(x_0,x_1) + v(1,y) \\ &\min_y \{V_0(x,y) + v(1,y)\} \leq V_0(x_0,x_1) + \sum_{t=1}^{T-1} V_t(x_T, x_{T+1}), \end{aligned} \tag{9}$$

d'où l'on obtient avec (8) que

$$\min_y \{V_0(x,y) + v(1,y)\} \leq \epsilon + V(0,x). \tag{10}$$

Pour terminer, on conclut alors que $V(0,x) = \min y \{V_0(x,y) + v(1,y)\}$.

□

Remarque. *Par récurrence on prouve le cas pour* $t \in [1; T-1]$ *dans le proposition précédente.*

4.1 Algorithmique

On calcul avec backward/rétrograde : Pour tout x

$$v(T-1,x) = \min_y \{V_T(x,y)\} \tag{11}$$

$$v(T-2,x) = \min_y \{v(T-1,x) + V_{T-2}(x,y)\} \tag{12}$$

$$\vdots \tag{13}$$

$$v(0,x) = \dots \tag{14}$$

Remarque. *C'est un algorithme seulement si l'on peut résoudre chacun des problèmes* $V_{T-1}(x,y)$.

Références

[Bel54] R. Bellman. The theory of dynamic programming. Bull. Amer. Math. Soc., 60(6) :503–515, 11 1954. 2

[Car07] G. Carlier. Programmation dynamique, 2007. 1