

La Programmation dynamique de Richard Bellman

Jill-Jênn Vie

8 mai 2015

Approche naïve

```
def fibo(n):  
    if n <= 1:  
        return n  
    else :  
        return fibo(n - 1)  
           + fibo(n - 2)
```

Programmation dynamique

```
def fibo2(n):  
    f = [0] * (n + 1)  
    f[1] = 1  
    for i in range(2, n + 1):  
        f[i] = f[i - 2]  
              + f[i - 1]  
    return f[n]
```

Énoncé

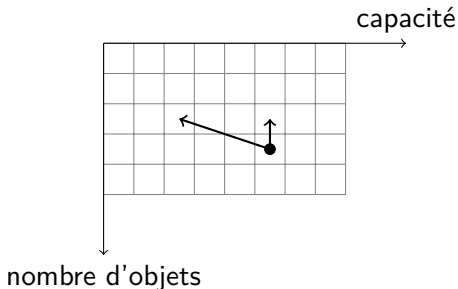
On dispose de n objets de capacités c_1, \dots, c_n et valeurs v_1, \dots, v_n . Étant donné un sac de capacité C , quel choix d'objets permet d'obtenir une valeur maximale ?

Résolution du sac à dos

Soit $\text{plusGrandeValeur}[i][c]$ la **plus grande valeur atteignable** avec les **i premiers objets** et une **capacité c** ($1 \leq i \leq n$, $1 \leq c \leq C$).

Pour le i -ième objet on a deux options :

- soit on le prend : $v_i + \text{plusGrandeValeur}[i-1][c - c_i]$
- soit on ne le prend pas : $\text{plusGrandeValeur}[i-1][c]$.



Définition

La programmation dynamique consiste à résoudre un problème d'optimisation en combinant des solutions à des sous-problèmes stockées au préalable (approche *bottom-up*)

Conditions

- **Sous-structure optimale** : une solution optimale peut être obtenue à partir des solutions optimales de certains sous-problèmes.
- Il existe **un ordre sur les sous-problèmes** compatible avec la sous-structure optimale.

Mais...

Non,

c'est la discrétisation de l'équation d'Hamilton-Jacobi-Bellman !

$$-\frac{\partial S(x, t)}{\partial t} = \min_{u \in \Omega} [G(x, u, t) + \nabla S(x, t) \cdot f(x, u, t)].$$

- Algorithmique
- Mathématiques : calcul des variations (EDP)
- Automatique : théorie du contrôle optimal
- Machine learning : apprentissage par renforcement
- Croissance économique
- Problèmes principal-agent
- Finances publiques
- Politique monétaire et budgétaire
- Théorie de la recherche d'emploi
- Réchauffement climatique
- Faim dans le monde

The great drawback of dynamic programming is, as Bellman himself calls it, the "curse of dimensionality." Even recording the solution to a moderately complicated problem involves an enormous amount of storage. If we want only one optimal path from a known initial point, it is wasteful and tedious to find a whole field of extremals; if we need feedback, a perturbation feedback scheme is often quite adequate (see Chapter 6, Neighboring Extremals and the Second Variation).

Derivation of the Euler-Lagrange equations from the Hamilton-Jacobi equation. Consider a particular optimal path and its associated optimal control function. Then we have

$$\frac{d\lambda^T}{dt} = \frac{d}{dt} \left(\frac{\partial f^0}{\partial \dot{x}} \right) = \frac{\partial^2 f^0}{\partial \dot{x}^2} \dot{x} + \frac{\partial^2 f^0}{\partial x \partial t} \quad (4.2.19)$$

Partial differentiation of (4.2.15) with respect to x , considering $u^0 = u^0(x, t)$, gives

$$\frac{\partial^2 f^0}{\partial x \partial t} + \frac{\partial L}{\partial x} + \frac{\partial L}{\partial u} \frac{\partial u^0}{\partial x} + f^T \frac{\partial^2 f^0}{\partial x^2} + \frac{\partial f^0}{\partial x} \left(\frac{\partial f}{\partial x} + \frac{\partial f}{\partial u} \frac{\partial u^0}{\partial x} \right) = 0 \quad (4.2.20)$$

Now the coefficient of $\partial u^0 / \partial x$ in (4.2.20) vanishes on an optimal path according to (4.2.17).[†] Using (4.2.20) in (4.2.19), then, we obtain

$$\frac{d\lambda^T}{dt} = -\lambda^T \frac{\partial f}{\partial x} - \frac{\partial L}{\partial x} \quad (4.2.21)$$

which, along with (4.2.17), are the Euler-Lagrange equations.

Furthermore, the fact that f^0 is equal to ϕ on $\psi = 0$ implies that there exists a vector ν such that

$$\frac{\partial f^0}{\partial x} \Big|_{f^0} = \left(\frac{\partial \phi}{\partial x} + \nu^T \frac{\partial \psi}{\partial x} \right) \Big|_{f^0} \triangleq \lambda^T(t) \quad (4.2.22)$$

In words, the change in cost due to admissible change in state (that is, $d\psi = 0$) is given by a linear combination of the gradient of ϕ with respect to state and the gradients of ψ (constraints) with respect to state (see Section 1.2).

Combinatorial problems. Dynamic programming is especially useful in solving multistage optimization problems in which there are only a small number of possible choices of the control at each

[†]When there are inequality constraints on the control variable, it can be shown (e.g., by defining a modified Hamiltonian as in Chapter 3 or see Kalman (1963) or Dreyfus (1965)) that the term $(L_x + f_x^T \nu)_x^0$ still vanishes.

stage and in which no derivative information is available. Consider a simple example with only two possible choices of control at each stage. We would like to find the path from A to B in Figure 4.2.1,

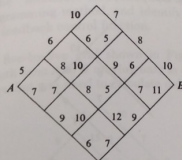


Figure 4.2.1. A minimum-time combinatorial problem. Numbers are times to travel legs of grid.

traveling only to the right (either up-right or down-right at each corner), such that the sum of the numbers on the segments along this path is a minimum. If we consider these numbers to be travel times, we are looking for the minimum-time path.

There are 20 possible routes from A to B, traveling only to the right. It would be tedious to try out all 20 routes. Instead of starting at A and trying different routes to B, we work *backward* from B to find the minimum-time routes to B from each of the 15 corners on the grid, as indicated on Figure 4.2.2.

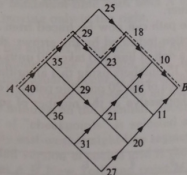


Figure 4.2.2. Dynamic programming solution to the problem of Figure 4.2.1.

The great drawback of dynamic programming is, as Bellman himself calls it, the "curse of dimensionality." Even recording the solution to a moderately complicated problem involves an enormous amount of storage. If we want only one optimal path from a known initial point, it is wasteful and tedious to find a whole field of extremals; if we need feedback, a perturbation feedback scheme is often quite adequate (see Chapter 6, Neighboring Extremals and the Second Variation).

Derivation of the Euler-Lagrange equations from the Hamilton-Jacobi equation. Consider a particular optimal path and its associated optimal control function. Then we have

$$\frac{d\lambda^T}{dt} = \frac{d}{dt} \left(\frac{\partial J^0}{\partial \dot{x}} \right) = \frac{\partial^2 J^0}{\partial \dot{x}^2} \dot{x} + \frac{\partial^2 J^0}{\partial x \partial t} \quad (4.2.19)$$

Partial differentiation of (4.2.15) with respect to x , considering $u^0 = u^0(x, t)$, gives

$$\frac{\partial^2 J^0}{\partial x \partial t} + \frac{\partial L}{\partial x} + \frac{\partial L}{\partial u} \frac{\partial u^0}{\partial x} + \dot{J}^T \frac{\partial^2 J^0}{\partial x^2} + \frac{\partial J^0}{\partial x} \left(\frac{\partial f}{\partial x} + \frac{\partial f}{\partial u} \frac{\partial u^0}{\partial x} \right) = 0 \quad (4.2.20)$$

Now the coefficient of $\partial u^0 / \partial x$ in (4.2.20) vanishes on an optimal path according to (4.2.17). Using (4.2.20) in (4.2.19), then, we obtain

$$\frac{d\lambda^T}{dt} = -\lambda^T \frac{\partial f}{\partial x} - \frac{\partial L}{\partial x} \quad (4.2.21)$$

which, along with (4.2.17), are the Euler-Lagrange equations.

Furthermore, the fact that J^0 is equal to ϕ on $\psi = 0$ implies that there exists a vector ν such that

$$\frac{\partial J^0}{\partial x} \Big|_{\psi} = \left(\frac{\partial \phi}{\partial x} + \nu^T \frac{\partial \psi}{\partial x} \right) \Big|_{\psi=0} \triangleq \lambda^T(t) \quad (4.2.22)$$

In words, the change in cost due to admissible change in state (that is, $d\psi = 0$) is given by a linear combination of the gradient of ϕ with respect to state and the gradients of ψ (constraints) with respect to state (see Section 1.2).

Combinatorial problems. Dynamic programming is especially useful in solving multistage optimization problems in which there are only a small number of possible choices of the control at each

¹When there are inequality constraints on the control variable, it can be shown (e.g., by defining a modified Hamiltonian as in Chapter 3 or see Kalman (1963) or Dreyfus (1965)) that the term $(L_x + J_x^T u_x^0)$ still vanishes.

stage and in which no derivative information is available. Consider a simple example with only two possible choices of control at each stage. We would like to find the path from A to B in Figure 4.2.1,

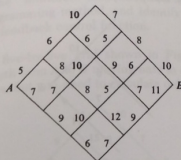


Figure 4.2.1. A minimum-time combinatorial problem. Numbers are times to travel legs of grid.

traveling only to the right (either up-right or down-right at each corner), such that the sum of the numbers on the segments along this path is a minimum. If we consider these numbers to be travel times, we are looking for the minimum-time path.

There are 20 possible routes from A to B, traveling only to the right. It would be tedious to try out all 20 routes. Instead of starting at A and trying different routes to B, we work backward from B to find the minimum-time routes to B from each of the 15 corners on the grid, as indicated on Figure 4.2.2.

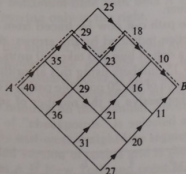


Figure 4.2.2. Dynamic programming solution to the problem of Figure 4.2.1.



- 1920 – 1984
- 1946 : thèse à Princeton
- 1949 : rejoint RAND, un *think tank* des US Army Air Forces
- 39 livres
- 619 articles de recherche publiés
- dont *Dynamic Programming and Lagrange Multipliers* cité 16 268 fois

- 1953 : premier langage de programmation (Autocode)
- 1953 : *An Introduction to the Theory of Dynamic Programming*
- 1954 : *The Theory of Dynamic Programming*
- 1957 : *Dynamic Programming*
- 1957 : FORTRAN
- 1958 : LISP
- 1968 : *The Art of Computer Programming* tome 1

Une politique optimale est telle que, quels que soient l'état initial et les décisions initiales, les décisions suivantes doivent constituer une politique optimale par rapport à l'état résultant des premières décisions.

Énoncé

Deux mines d'or, l'une en Anaconda (quantité de minerai x), l'autre en Bonanza (quantité y).

- Anaconda : probabilité p_1 de récolter r_1x $1 - p_1$ de FAIL.
- Bonanza : probabilité q_1 de récolter r_2y $1 - q_1$ de FAIL.

Quelle est l'action qui maximise la quantité espérée d'or ?

Soit $f(x, y)$ l'espérance de la quantité d'or extraite avant FAIL.

Si on choisit A : gain espéré $\rightarrow p_1(r_1x + f(x - r_1x, y))$

Si on choisit B : gain espéré $\rightarrow q_1(r_2y + f(x, y - r_2y))$

Il faut donc résoudre l'équation :

$$f(x, y) = \max \left\{ \begin{array}{l} p_1(r_1x + f((1 - r_1)x, y)) \\ q_1(r_2y + f(x, (1 - r_2)y)) \end{array} \right\}.$$

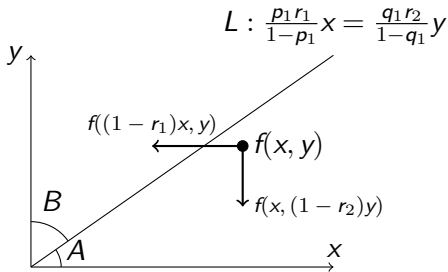
Solution

$$\text{Si } \frac{p_1 r_1}{1 - p_1} x > \frac{q_1 r_2}{1 - q_1} y,$$

Choisir A

Sinon

Choisir B.



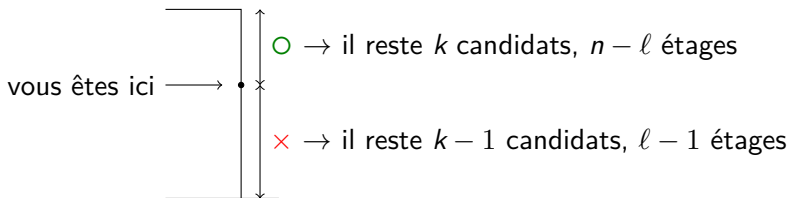
Énoncé

On dispose de k candidats Prologin et d'un immeuble à n étages. Pour déterminer l'étage critique, on jette des candidats par la fenêtre. (Si le candidat survit, on peut s'en resservir, sinon on ne peut pas.) En combien de sauts peut-on être sûr de connaître l'étage critique ?

Un algorithme pour $k = 2$

On répartit les étages en \sqrt{n} paquets de \sqrt{n} .
On envoie un candidat aux étages : $\sqrt{n}, 2\sqrt{n}, \dots$
Puis l'autre aux étages : $\ell\sqrt{n} + 1, \ell\sqrt{n} + 2, \dots$
Complexité : $\lfloor n/\sqrt{n} \rfloor + \sqrt{n} - 1 = O(\sqrt{n})$.

Soit $\text{nbMinSauts}[k][n]$ le nombre minimal de sauts à effectuer avec k candidats pour un bâtiment de n étages.



Il faut donc résoudre l'équation fonctionnelle :

$$\text{nbMinSauts}[k][n] = \min_{1 \leq \ell \leq n} \max \left\{ \begin{array}{l} \text{nbMinSauts}[k][n - \ell] \\ \text{nbMinSauts}[k - 1][\ell - 1] \end{array} \right\}.$$

... D'où vient le nom « programmation dynamique » ?

Exercices laissés au spectateur

- Max le Texan (2007)
- Urgence réseau (2008)

À résoudre sur prologin.org.