# La Programmation dynamique de Richard Bellman

JJ Vie

# Nombres de Fibonacci

## 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]
```

# Problème du sac à dos

## Énoncé

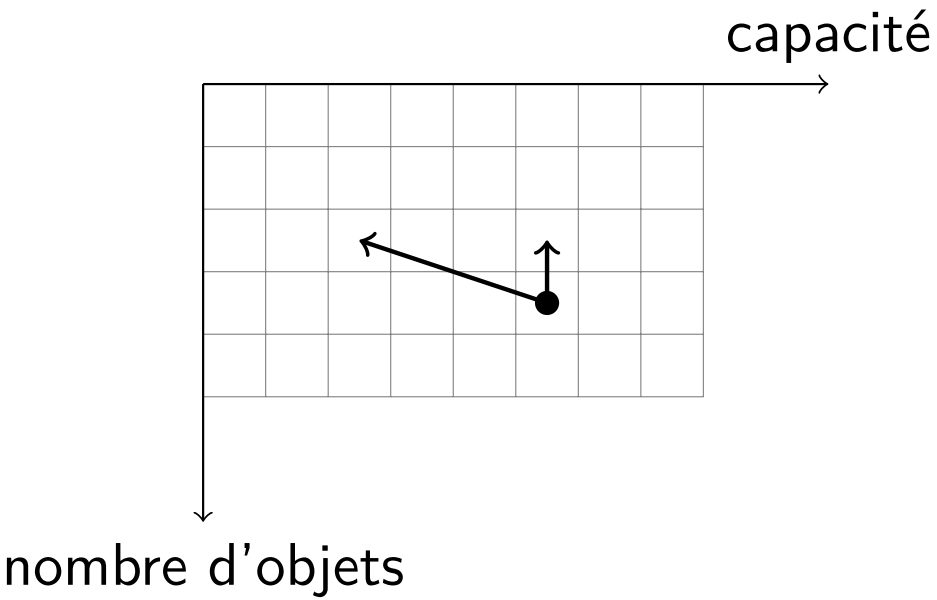
On dispose de $n$ objets de capacités $c_1, \ldots, c_n$ et valeurs $v_1, \ldots, v_n$. Étant donné un sac de capacité $C$, quel choix d'objets permet d'obtenir une valeur maximale ?

# Résolution du sac à dos

Soit 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

## 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.

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

$$\frac{d\lambda^T}{dt} \equiv \frac{d}{dt}\left(\frac{\partial J^\circ}{\partial x}\right) = \frac{\partial^2 J^\circ}{\partial x^2}\dot{x} + \frac{\partial^2 J^\circ}{\partial x \partial t}. \tag{4.2.19}$$

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

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

Now the coefficient of $\partial u^\circ/\partial x$ 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

$$\frac{d\lambda^T}{dt} = -\lambda^T\frac{\partial f}{\partial x} - \frac{\partial L}{\partial x}, \tag{4.2.21}$$

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

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

$$\left.\frac{\partial J^\circ}{\partial x}\right|_{t_f} = \left(\frac{\partial \phi}{\partial x} + \nu^T\frac{\partial \psi}{\partial x}\right)_{t=t_f} \triangleq \lambda^T(t_f). \tag{4.2.22}$$

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 $\phi$ with respect to state and the gradients of $\psi$ (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 $(L_u + J_x^\circ f_u)u_x^\circ$ 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, 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.

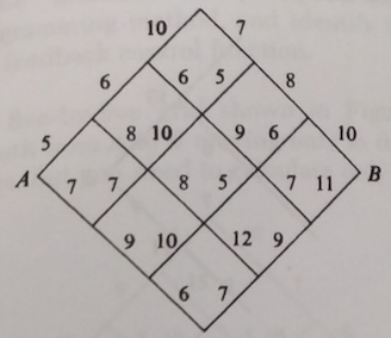



**Figure 4.2.1.** A minimum-time combinatorial problem. Numbers are times to travel legs of grid.

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.

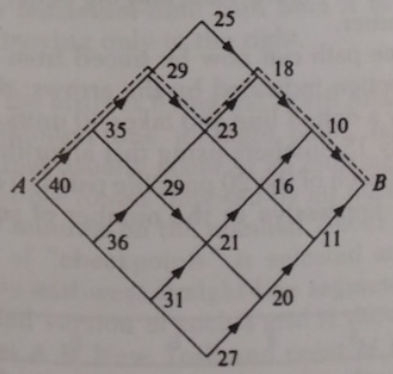



**Figure 4.2.2.** Dynamic programming solution to the problem of Figure 4.2.1.

# Richard Bellman

- 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

# Quelques dates

- 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

# Politique optimale

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.

# Problème des mines d'or

## 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 $r_1 x$ $\quad$ $1 - p_1$ de FAIL.
- Bonanza : probabilité $q_1$ de récolter $r_2 y$ $\quad$ $1 - q_1$ 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 : $\quad$ gain espéré $\rightarrow p_1(r_1 x + f(x - r_1 x, \quad y \quad))$
Si on choisit B : $\quad$ gain espéré $\rightarrow q_1(r_2 y + f(\quad x \quad, y - r_2 y))$

Il faut donc résoudre l'équation :

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

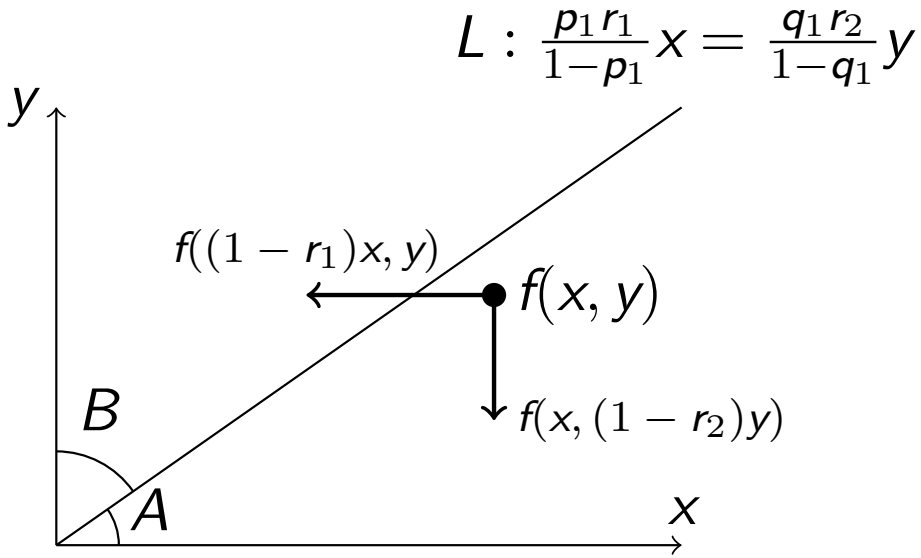
**Solution**

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

Choisir $A$

Sinon

Choisir $B$.



# Un autre problème

## Énoncé

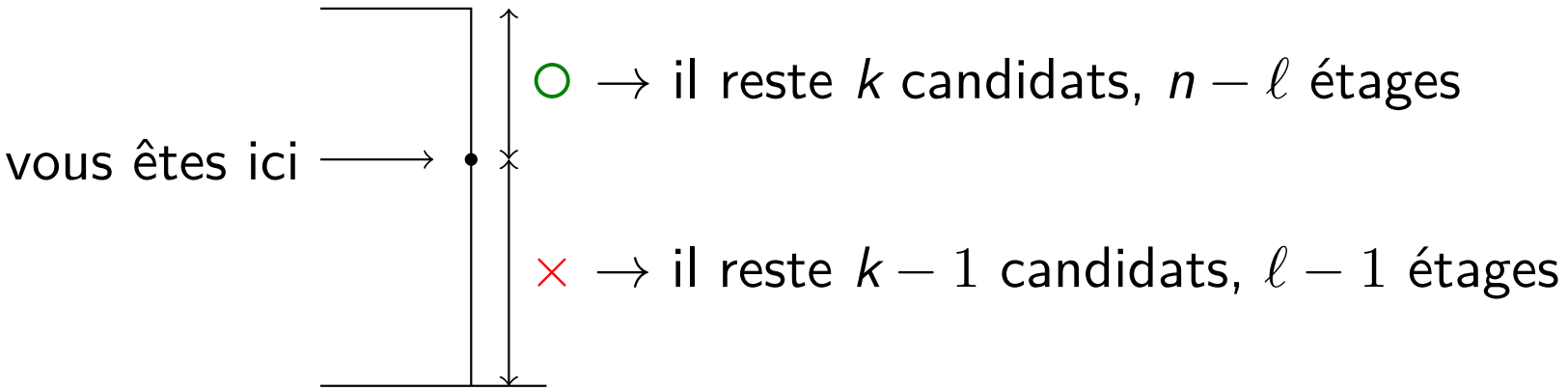
On dispose de $k$ candidats Prologin et d'un immeuble à $n$ étages.
Pour déterminer l'étage critique, on jette des candidats par la fenêtre. (Si le candidat survit, on peut s'en resservir, sinon on ne peut pas.)
En combien de sauts peut-on être sûr de connaître l'étage critique ?

## Un algorithme pour $k = 2$

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

# Résolution

Soit 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 :

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

# Une dernière question

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