

Programowanie dynamiczne

Programowanie dynamiczne

- Równania Bellmana można rozwiązać analitycznie.
- Złożoność obliczeniowa sięga $O(n^3)$ gdzie n oznacza liczbę stanów.
- W przypadku rozwiązywania dużych problemów istnieje wiele metod iteracyjnych:
 - Exhaustive search
 - Programowanie dynamiczne
 - Metody Monte Carlo
 - Temporal Difference Learning

Programowanie dynamiczne

- Dużo efektywniejszym sposobem rozwiązywania procesów decyzyjnych Markowa jest wykorzystanie metod programowania dynamicznego.
- Jest to możliwe dzięki strukturze procesów decyzyjnych Markowa:
 - Rozwiązanie można rozbić na zagadnienie optymalizacji podproblemów (**zasada optymalności Bellmana**).
 - Podproblemy powtarzają się – to samo rozwiązanie można wykorzystać wiele razy.

Programowanie dynamiczne

- Dużo efektywniejszym sposobem rozwiązywania procesów decyzyjnych Markowa jest wykorzystanie metod programowania dynamicznego.
- Jest to możliwe dzięki strukturze procesów decyzyjnych Markowa:
 - Rozwiązanie można rozbić na zagadnienie optymalizacji podproblemów (**zasada optymalności Bellmana**).
 - Podproblemy powtarzają się – to samo rozwiązanie można wykorzystać wiele razy.
- **Zasada optymalności Bellmana:** optymalna strategia ma tę własność, że niezależnie od warunków początkowych i początkowej decyzji, akcje podjęte w każdej iteracji muszą stanowić optymalną strategię dla stanów wynikłych z poprzednich decyzji.

Programowanie dynamiczne

- **Ewaluacja** – dana jest strategia π , należy dokonać jej oceny:
 - Zainicjuj algorytm wybierając dowolne początkowe wartości funkcji wartości, np. $V_0(s) = 0 \ \forall s$.
 - W każdym kroku k :
 - dla każdego ze stanów $s \in S$ uaktualnij V_k na podstawie $V_{k-1}(s')$:

$$V_k(s) = \sum_a \pi(s, a) \sum_{s', r} P(s, a, s') (r + \beta V_{k-1}(s'))$$

- Zatrzymaj proces gdy:

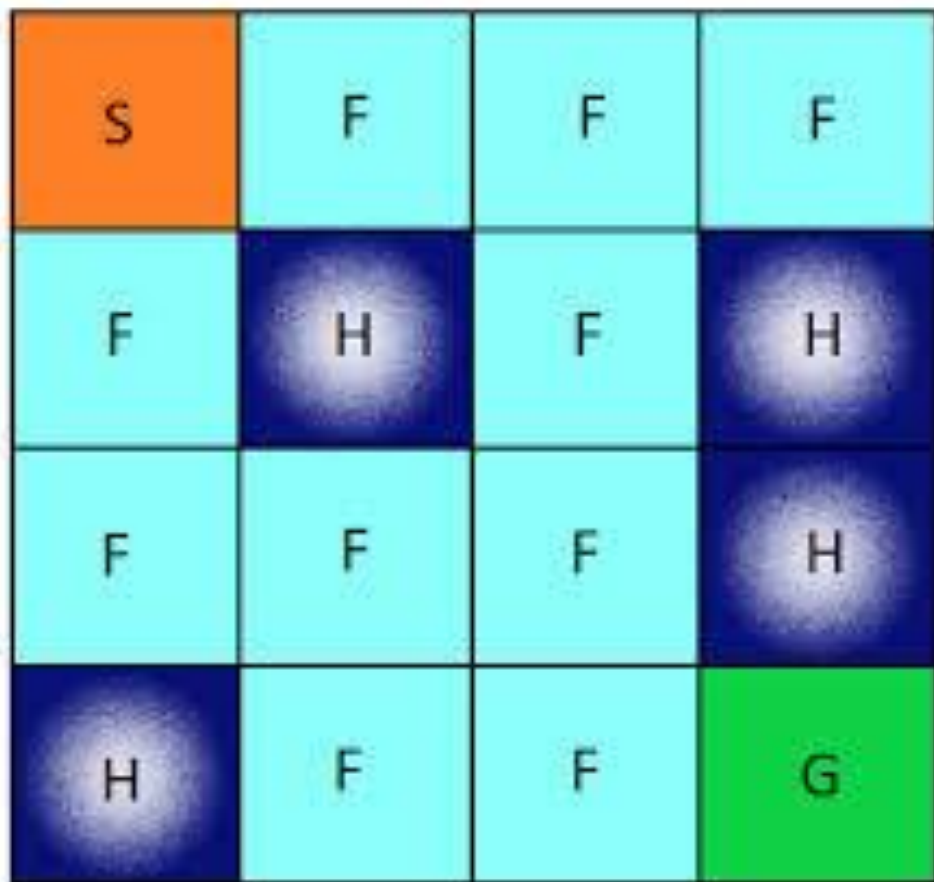
$$\max_s |V_k(s) - V_{k-1}(s)| < \theta$$

Programowanie dynamiczne

- **Ewaluacja** – dana jest strategia π , należy dokonać jej oceny:
 - Zainicjuj algorytm wybierając dowolne początkowe wartości funkcji wartości, np. $V_0(s) = 0 \ \forall s$.
 - W każdym kroku k :
 - dla każdego ze stanów $s \in S$ uaktualnij V_k na podstawie $V_{k-1}(s')$:
$$V_k(s) = \sum_a \pi(s, a) \sum_{s', r} P(s, a, s') (r + \beta V_{k-1}(s'))$$
 - Zatrzymaj proces gdy:
$$\max_s |V_k(s) - V_{k-1}(s)| < \theta$$
- Na mocy **Twierdzenia Banacha o punkcie stałym** algorytm zbiega do V_π .

Przykład

Frozen Lake



A 4x4 grid representing the Frozen Lake environment. The start state 'S' is in the top-left cell (orange). The goal state 'G' is in the bottom-right cell (green). Ice blocks are represented by light blue cells labeled 'F'. Holes are represented by dark blue cells labeled 'H'. The grid layout is as follows:

S	F	F	F
F	H	F	H
F	F	F	H
H	F	F	G

- Celem agenta jest przejść z punktu **S** do Punktu **G**.
- Agent może iść po lodzie (pola oznaczone literą **F**), musi unikać wpadnięcia do przerębli (pola oznaczone jako **H**).
- Lód jest śliski; idąc przed siebie z pewnym prawdopodobieństwem p może się poślizgnąć i przesunąć w lewo lub w prawo w stosunku do swojej wyjściowej pozycji.

Programowanie dynamiczne

- **Optymalizacja strategii** – poszukiwanie optymalnej strategii π_* i optymalnej funkcji wartości stanu V_* .
- Metody rozwiązania:
 - Iterowanie po funkcji wartości (**value iteration**).
 - Iterowanie po strategii (**policy iteration**).
 - Metody hybrydowe (**metoda aktora – krytyka**).

Value iteration

1. Zainicjuj algorytm wybierając dowolne początkowe wartości funkcji wartości, np. $V_0(s) = 0 \forall s$.
2. W każdej iteracji k :
 - Uaktualnij wartość $V_k(s)$ wykorzystując do tego estymację funkcji $V_{k-1}(s')$:

$$V_k(s) = \max_a \sum_{s',r} P(s, a, s')(r + \beta V_{k-1}(s'))$$

3. Zatrzymaj algorytm gdy wynik zbiegnie do $\max_s |V_k(s) - V_{k-1}(s)| < \theta$.
4. W ostatnim kroku wygeneruj optymalną strategię π_* wykorzystując do tego równanie:

$$\pi_* = \operatorname{argmax}_a \sum_{s,r} P(s, a, s')(r + V_{\pi_*}(s'))$$

Policy iteration

1. Zainicjuj algorytm wybierając dowolne początkowe wartości funkcji wartości, np. $V_0(s) = 0 \forall s$ i dowolną strategię π .

2. W każdej iteracji k :

- Dokonaj oceny $V(s)$ wykorzystując do tego strategię π :

$$V_k(s) = \sum_a \pi(s, a) \sum_{s', r} P(s, a, s') (r + \beta V_\pi(s'))$$

- Popraw zaproponowaną strategię zachowując się zachłannie w stosunku do V_π :

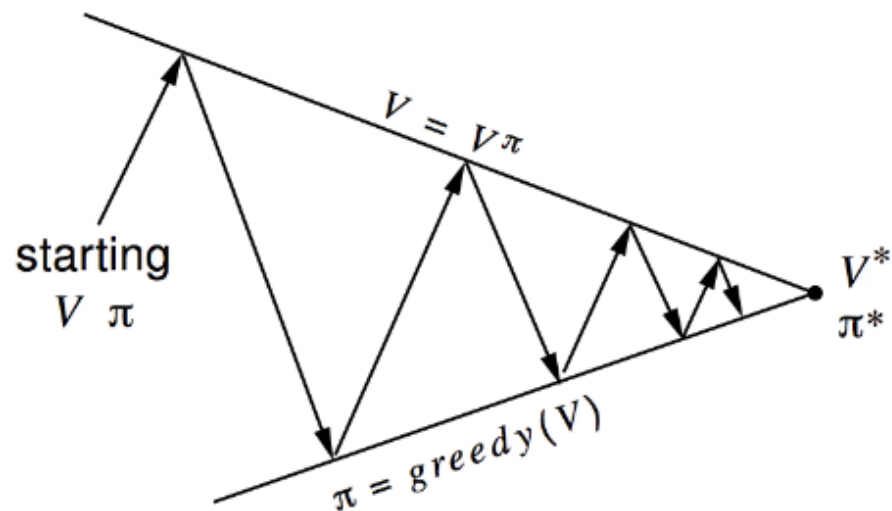
$$\pi' = \text{greedy}(V_\pi)$$

$$\pi' = \operatorname{argmax}_a \sum_{s, r} P(s, a, s') (r + V_\pi(s'))$$

3. Zatrzymaj algorytm gdy wynik zbiegnie do $\max_s |V_k(s) - V_{k-1}(s)| < \theta$.

4. Otrzymana w ten sposób strategia π' jest optymalna: $\pi' = \pi_*$.

Policy iteration

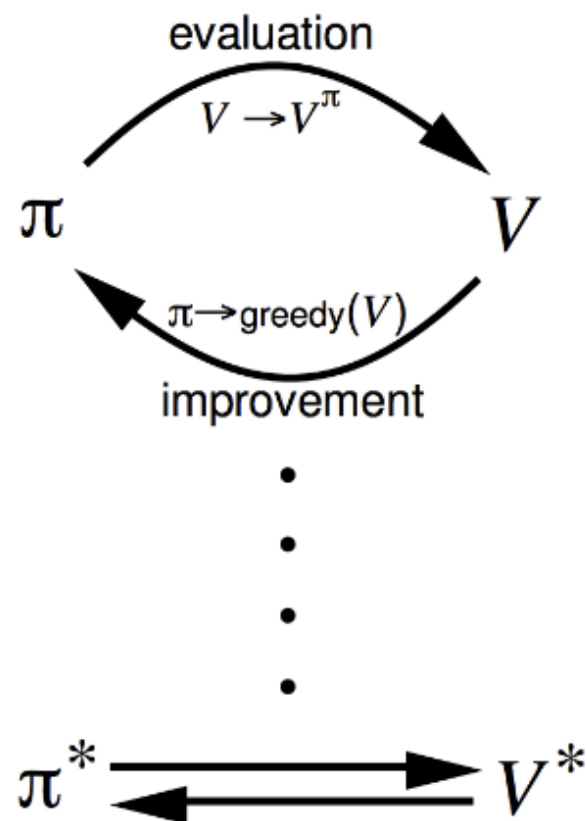


Policy evaluation Estimate v_π

Iterative policy evaluation

Policy improvement Generate $\pi' \geq \pi$

Greedy policy improvement

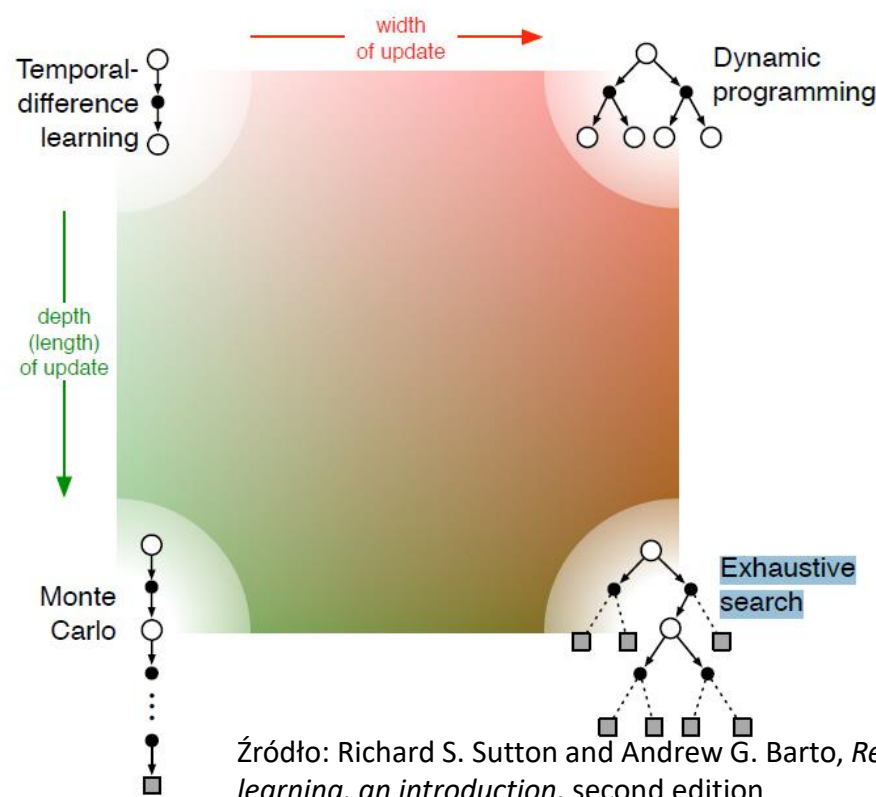


Programowanie dynamiczne

- Metody programowania dynamicznego gwarantują zbieżność do rozwiązania optymalnego.
- Jednak nadal charakteryzują się dużą złożonością obliczeniową, przez którą rozwiązywanie dużych problemów może być trudne.

Programowanie dynamiczne

- Metody programowania dynamicznego gwarantują zbieżność do rozwiązania optymalnego.
- Jednak nadal charakteryzują się dużą złożonością obliczeniową, przez którą rozwiązywanie dużych problemów może być trudne:



Programowanie dynamiczne

- Metody programowania dynamicznego gwarantują zbieżność do rozwiązania optymalnego.
- Jednak nadal charakteryzują się dużą złożonością obliczeniową, przez którą rozwiązywanie dużych problemów może być trudne.
- Można temu zaradzić poprzez rezygnację z *synchronicznych backupów* na rzecz *asynchronicznych*:
 - in-place dynamic programming
 - Prioritised sweeping
 - Real-time dynamic programming

Programowanie dynamiczne

- Przy czym nadal do poprawnego działania tych algorytmów konieczna jest pełna wiedza na temat rozpatrywanego procesu decyzyjnego Markowa (nazywamy to uczeniem opartym na modelu – *model based learning*).
- Konieczna jest znajomość macierzy przejścia i nagród za znalezienie się w każdym z osiągalnych stanów.
- Nie zawsze taka wiedza jest dostępna – wtedy warto skorzystać z metod uczenia wolnego od modeli (*model free learning*).