Algoritmen & Datastructuren 2

WPO – H13 – Dynamisch Programmeren

(Ruwe) Samenvatting HOC

Dynamisch "Programmeren" = strategie om optimalisatieproblemen efficiënt aan te pakken

1. Structuur van optimale oplossingen karakteriseren
2. Optimale waarde recursief definiëren
3. Optimale waarde berekenen
    a. Top-down met memoize
    b. **Bottom-up met tabulatie = Dynamisch Programmeren**
4. Optimale oplossing extraheren

Simpel voorbeeld: Fibonacci

- Overlapping tussen berekening `fib(n-1)` en `fib(n-2)`
- Boomrecursie = meest simpele manier om de oplossing te implementeren
- **Exponentieel verloop en geheugenverbruik (backtracking)!**

```
(define (fib n)
  (if (< n 2)
    1
    (+ (fib (- n 1))
       (fib (- n 2)))))
```

Fibonacci memoize (top-down)

- Start bij het grote probleem en breek die op in deelproblemen
- Gebruik data(structuur) voor deeloplossingen te bewaren = tabulatie
- Lichte verbetering in runtime, maar **blijft exponentieel door boomrecursie**

```
(define (fib n)
  (define fib-tab (make-vector (+ n 1) #f)
  (define (fib-rec n)
    (if (not (vector-ref fib-tab n))
      (vector-set! fib-tab n (+ (fib-rec (- n 1))
                                (fib-rec (- n 2))))))
    (vector-ref fib-tab n))
  (vector-set! fib-tab 0 1)
  (vector-set! fib-tab 1 1)
  (fib-rec n))
```

Fibonacci à la Dynamic Programming (bottom-up)

- Gebruik data(structuur) voor deeloplossingen te bewaren = **tabulatie**
- Los eerst kleine deelproblemen op die bijdragen aan het grotere op te lossen probleem
- **Iteratief**

```
(define (fib n)
  (define (iter n a b)
    (if (= n 0)
      b
      (iter (- n 1) b (+ a b))))
  (iter n 0 1))
```

Grafen: Transitieve sluiting / All Pair Shortest Path

Algoritmes van *Warshall* en *Floyd-Warshall* zijn beide voorbeelden van Dynamisch Programmeren

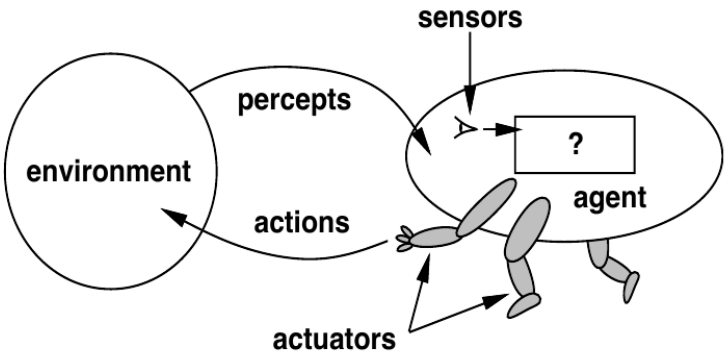
- Gelijkaardige uiteenzetting zoals Fibonacci is terug te vinden in
  - `a-d/graph-algorithms/directed/traclo-unweighted.rkt`
  - `a-d/graph-algorithms/directed/traclo-weighted.rkt`
- Idee: pad van knoop **`from`** naar **`to`** bestaat uit deelpad(en) langs mogelijke tussenkn(o)op(en), **`via`**
  - Mogelijke deelpaden zoeken en combineren
    - **`from`** ➔ **`via`**
    - **`via`** ➔ **`to`**

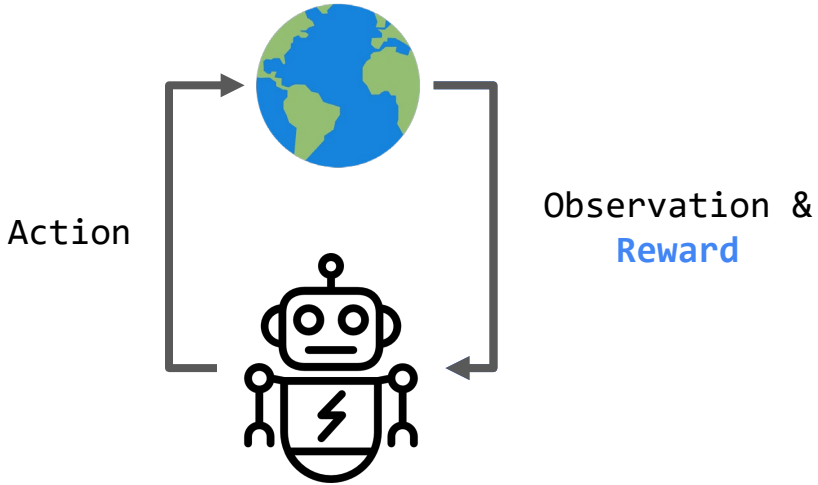
WPO Vandaag: Value Iteration in een 2D gridworld

- Algoritme ontwikkeld door Richard E. Bellman
  - Bedenker van Dynamic Programming
  - Dezelfde Bellman als in algoritme van Bellman-Ford!
- 1957
- Basis grondbeginselen van Reinforcement Learning

Beetje Context: Reinforcement Learning (RL)

- Agent = systeem dat een omgeving kan waarnemen en erin ageert
  - Sensoren en actuatoren
  - Fundamenteel begrip binnen de Artificiële Intelligentie
- Reinforcement Learning = techniek om agent te laten leren over hun omgeving/doel door middel van **feedback**
  - **Leer beloning te maximaliseren**





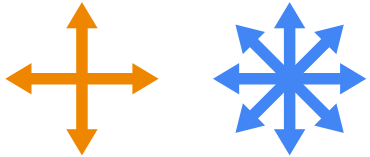
Beetje Context: Markov Decision Process (MDP)

Wiskundig formalisme om een RL omgeving voor te stellen

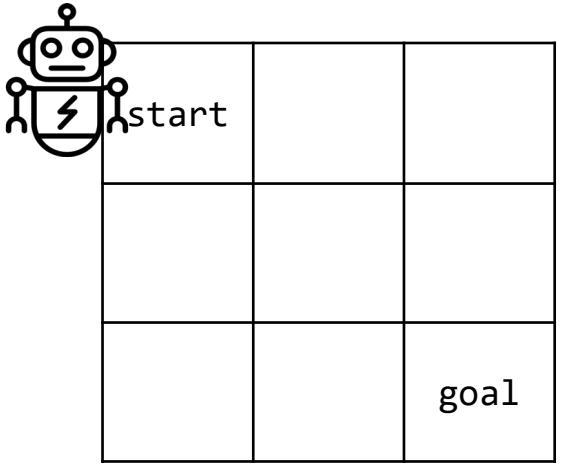
- S: Verzameling **toestanden** van de wereld
- A: Verzameling van **acties** die ondernomen kunnen worden in de wereld
- $\mathrm{p}(s, a, s')$: $S \times A \times S \mapsto \mathbb{R}$: Transitiefunctie die de **dynamiek** van de omgeving definieert
  - Mogelijke overgangen van toestand naar toestand door middel van acties **uitgedrukt in kans**
  - Alternatieve formulering $\mathrm{P}(s, a)$: $S \times A \mapsto \mathrm{S}$, de toestand waar je terecht komt na een actie te hebben uitgevoerd in een gegeven toestand
- $\mathrm{r}(s, a, s')$: $S \times A \times S \mapsto \mathbb{R}$: **Beloning**sfunctie, bepaalt of de agent iets goeds heeft gedaan
- $\gamma \in [0, 1]$: Discount factor (niet heel belangrijk voor vandaag)

Een gridworld als Markov Decision Process

- S: alle mogelijke coördinaten (x, y) in het grid
- A: alle mogelijke looprichtingen
  - Manhattan moves
  - King's moves



- $\mathrm{p}(s, a)$: het coördinaat waar je terecht komt na in een bepaalde richting een stap te hebben gezet (deterministisch)
- $\mathrm{r}(s, a, s')$: een positief getal als je het doel bereikt



Beetje Context: Value Function

Value Function $v(s)$ bepaalt hoe goed/slecht het is om in een bepaalde toestand te zijn

- Hoe? Bereken hoeveel reward er **kan** gehaald worden op termijn vanuit die toestand.
- Wat is de optimale value function, $v_*(s)$?

$$v_*(s) = \max_a \mathbb{E}[R_{t+1} + \gamma v_*(s_{t+1}) \mid s_t = s, a_t = a]$$

$$= \max_a \sum_{s',r} p(s', r \mid s, a) \, [r + \gamma v_*(s')]$$

Beetje Context: Policy

Welke actie is **de beste** om nu (in deze toestand) uit te voeren?

- Functie $\pi(s)$: $S \mapsto A$ bepaalt welke actie moet worden uitgevoerd.
- Kies een actie die je telkens naar de naburige toestand brengt met de hoogste value!
  - Indien de gebruikte value function optimaal is ($v_*(s)$), dan is deze policy optimaal.

$$\pi(s) = \operatorname{argmax}_a \sum_{s', r} p(s', r \mid s, a)\,[r + \gamma v(s')]$$

Value Iteration

- Benaderingsmethode voor het berekenen van de optimale value function $v_*(s)$ d.m.v. het opstellen van een tabel
- Voor iedere toestand in de omgeving, update de value iteratief tot een (bijna) fixpunt

Value Iteration, for estimating $\pi \approx \pi_*$

Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

Loop:
| $\Delta \leftarrow 0$
| Loop for each $s \in \mathcal{S}$:
| $v \leftarrow V(s)$
| $V(s) \leftarrow \max_a \sum_{s',r} p(s', r \mid s, a)\big[r + \gamma V(s')\big]$
| $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$

Output a deterministic policy, $\pi \approx \pi_*$, such that
$\pi(s) = \text{argmax}_a \sum_{s',r} p(s', r \mid s, a)\big[r + \gamma V(s')\big]$

Opdracht

- Bestudeer code in het bestand **`value-iteration.rkt`**
  - Voornamelijk procedures:
    - `v*-very-naive` (simpele boomrecursie)
    - `v*-naive` (top-down memoize)
  - Eventueel ook het bestand **`gridworld-mdp.rkt`**
- Vervolledig de body van de procedure `value-iteration`
  - Bottom-up + tabulatie
- Run het bestand om te zien wat de agent doet in de verschillende aangemaakte omgevingen