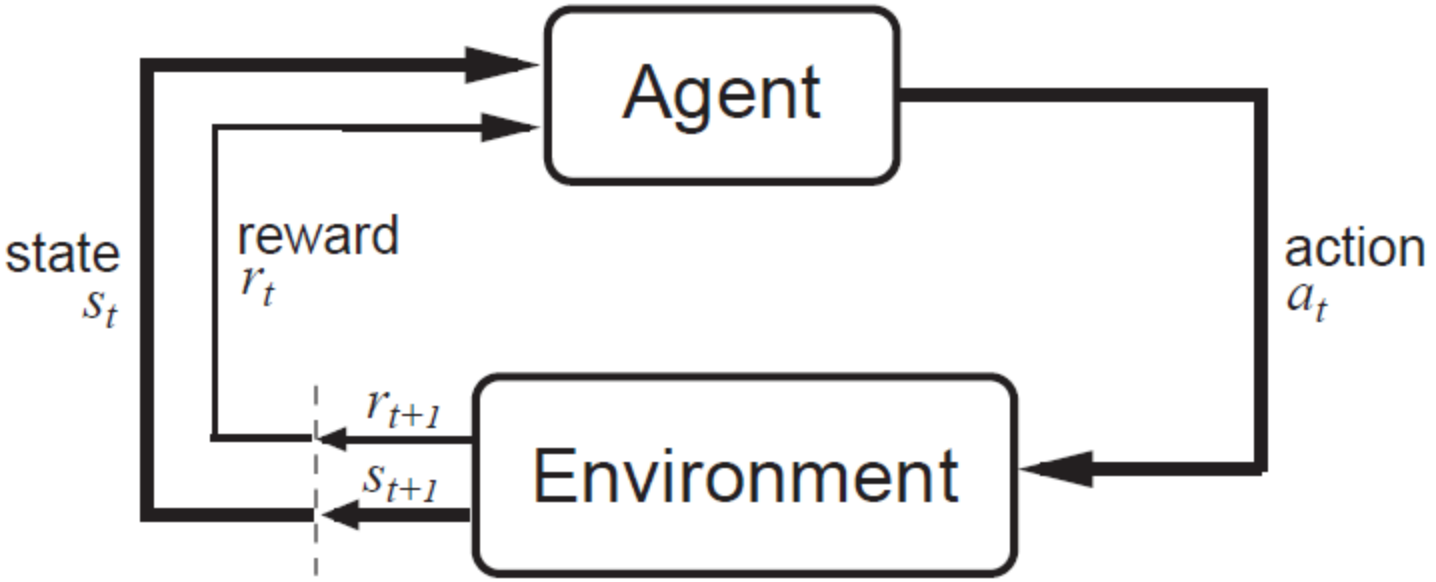
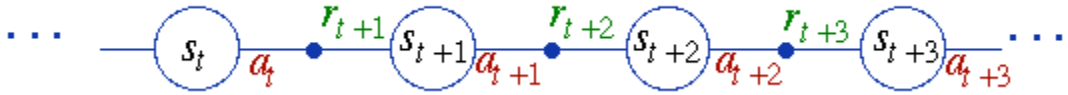
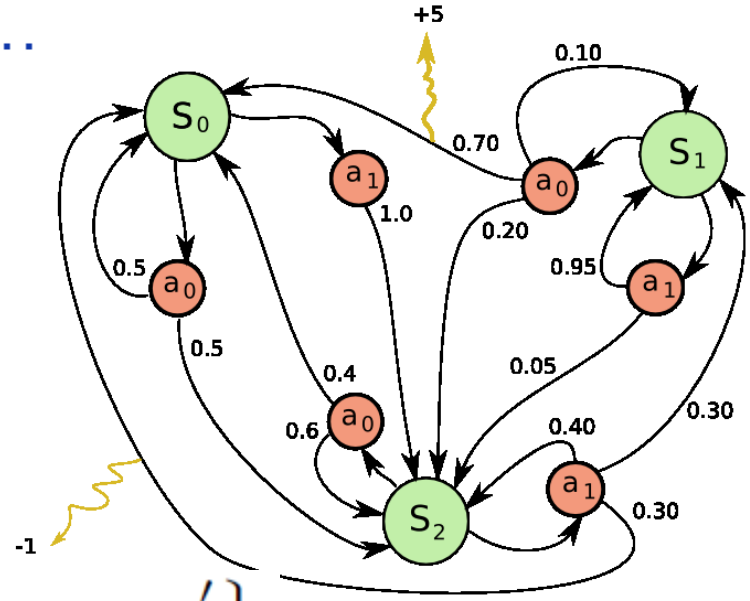
# Uczenie ze wzmocnieniem

# O co chodzi w uczeniu ze wzmocnieniem ?



# Proces decyzyjny Markowa (MDP)




- S – zbiór stanów
- A(s) – zbiór akcji dla każdego stanu s
- Nagroda:

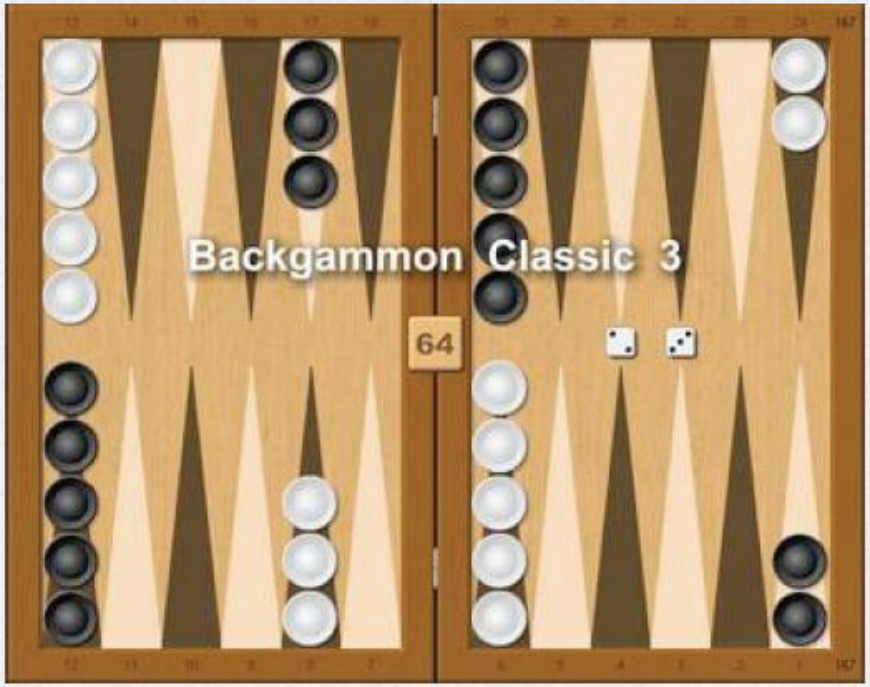
$$r^a_{ss'} = E\left\{r_{t+1} | s_t = s, a_t = a, s_{t+1} = s'\right\}$$

- Pra-wo przejścia:

$$p^a_{ss'} = P\left(s_{t+1} = s' | s_t = s, a_t = a\right)$$

**Warunek Markowa**: wartości $s_{t+1}$ i $r_{t+1}$ zależą jedynie od $s_t, a_t$ i nic poza tym.

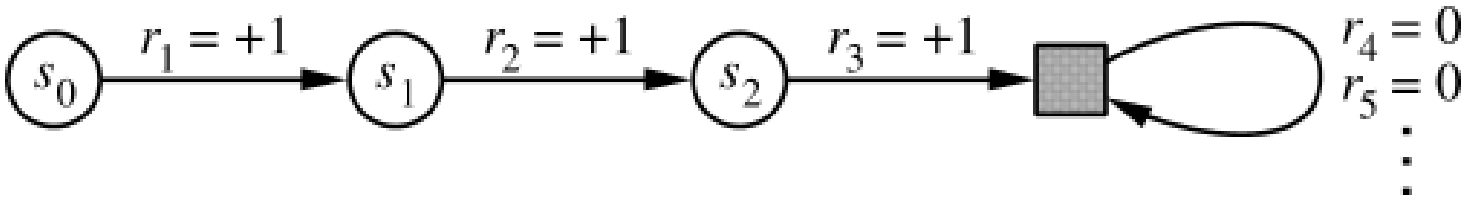
# Przykład




- **States:** board configurations (about $10^{20}$)
- **Actions:** permissible moves
- **Rewards:** win +1, lose -1, else 0

# Problem uczenia się dla agenta:

- Znaleźć stretegię $\pi : S \times A \rightarrow [0, 1]$, gdzie $\pi(s, a) = P(a_t = a | s_t = s)$ to prawdopodobieństwo wyboru akcji *a* w stanie *s*
- Zadania uczenia się dzielimy na
  - Epizodyczne (wyróżniamy stany końcowe)
    $R_t = r_{t+1} + r_{t+2} + \cdots + r_T$
  - Ciągłe.




$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots = \sum_{k=1}^{\infty} \gamma^{t+k-1} r_{t+k}$$

# Funkcje celu

- Definiujemy funkcję wartości stanu

$$V^{\pi}(s) = E_{\pi}\{R_t \mid s_t = s\} = E_{\pi}\left\{\sum_{k=1}^{\infty} \gamma^{k-1} r_{t+k} \mid s_t = s\right\}$$

- Funkcję wartości stanu-akcji

$$Q^{\pi}(s,a) = E_{\pi}\left\{\sum_{k=1}^{\infty} \gamma^{k-1} r_{t+k} \mid s_t = s, a_t = a\right\}$$

# Optymalna strategia

- Możemy definiować porządek częściowy miedzy strategiami

$$\pi \geq \pi' \iff V^{\pi}(s) \geq V^{\pi'}(s), \forall s \in S$$

- Bellman udowodnił (1950), że istnieje jedna i tylko jedna wartość optymalna dla każdego stanu

$$V^*(s) = \max_{\pi} V^{\pi}(s)$$

- Oraz istnieje conajmniej jedna strategia optymalna

$$\pi^* = \arg\max_{\pi} V^{\pi}$$

# Równania Bellmana dla funkcji wartości stanu

$$
\begin{aligned}
V^{\pi}(s) &= E_{\pi}\left\{ r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots \mid s_t = s \right\} \\
&= E_{\pi}\left\{ r_{t+1} + \gamma V(s_{t+1}) \mid s_t = s \right\} \\
&= \sum_{a \in A} \pi(s,a) \sum_{s' \in S} p^{a}_{ss'} \left( r^{a}_{ss'} + \gamma V^{\pi}(s') \right)
\end{aligned}
$$

Jest to układ równań liniowych, którego jedynym rozwiązaniem jest $V^{\pi}$

**Optymalizacyjne równania Bellmana dla optymalnej strategii**

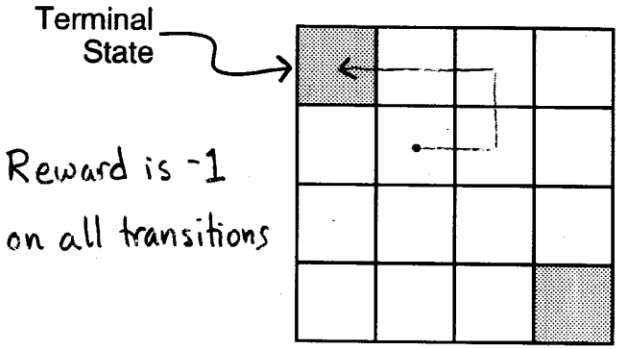
$$
V^{*}(s) = \max_{a \in A} \sum_{s' \in S} p^{a}_{ss'} \left( r^{a}_{ss'} + \gamma V^{*}(s') \right)
$$

# Równania optymalności Bellmana

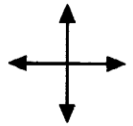
$$
\begin{aligned}
V^*(s) &= \max_{a \in \mathcal{A}(s)} Q^{\pi^*}(s,a) \\
&= \max_a E_{\pi^*}\left\{ R_t \;\middle|\; s_t = s, a_t = a \right\} \\
&= \max_a E_{\pi^*}\left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \;\middle|\; s_t = s, a_t = a \right\} \\
&= \max_a E\left\{ r_{t+1} + \gamma V^*(s_{t+1}) \mid s_t = s, a_t = a \right\} \\
&= \max_{a \in \mathcal{A}(s)} \sum_{s'} \mathcal{P}^a_{ss'} \left[ \mathcal{R}^a_{ss'} + \gamma V^*(s') \right].
\end{aligned}
$$

$$
\begin{aligned}
Q^*(s,a) &= E\left\{ r_{t+1} + \gamma \max_{a'} Q^*(s_{t+1}, a') \;\middle|\; s_t = s, a_t = a \right\} \\
&= \sum_{s'} \mathcal{P}^a_{ss'} \left[ \mathcal{R}^a_{ss'} + \gamma \max_{a'} Q^*(s', a') \right].
\end{aligned}
$$

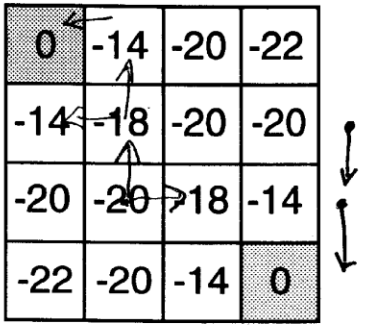
# Example of Value Functions




A Grid-Environment



Actions




State-Value Function for the random policy

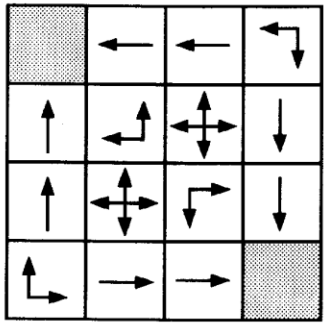
$V^{\pi}$

$Q^{\pi}(s,a)$

$V^*$

| | | | |
|---|---|---|---|
| 0 | -1 | -2 | -3 |
| -1 | -2 | -3 | -2 |
| -2 | -3 | -2 | -1 |
| -3 | -2 | -1 | 0 |

Optimal State-Value Function

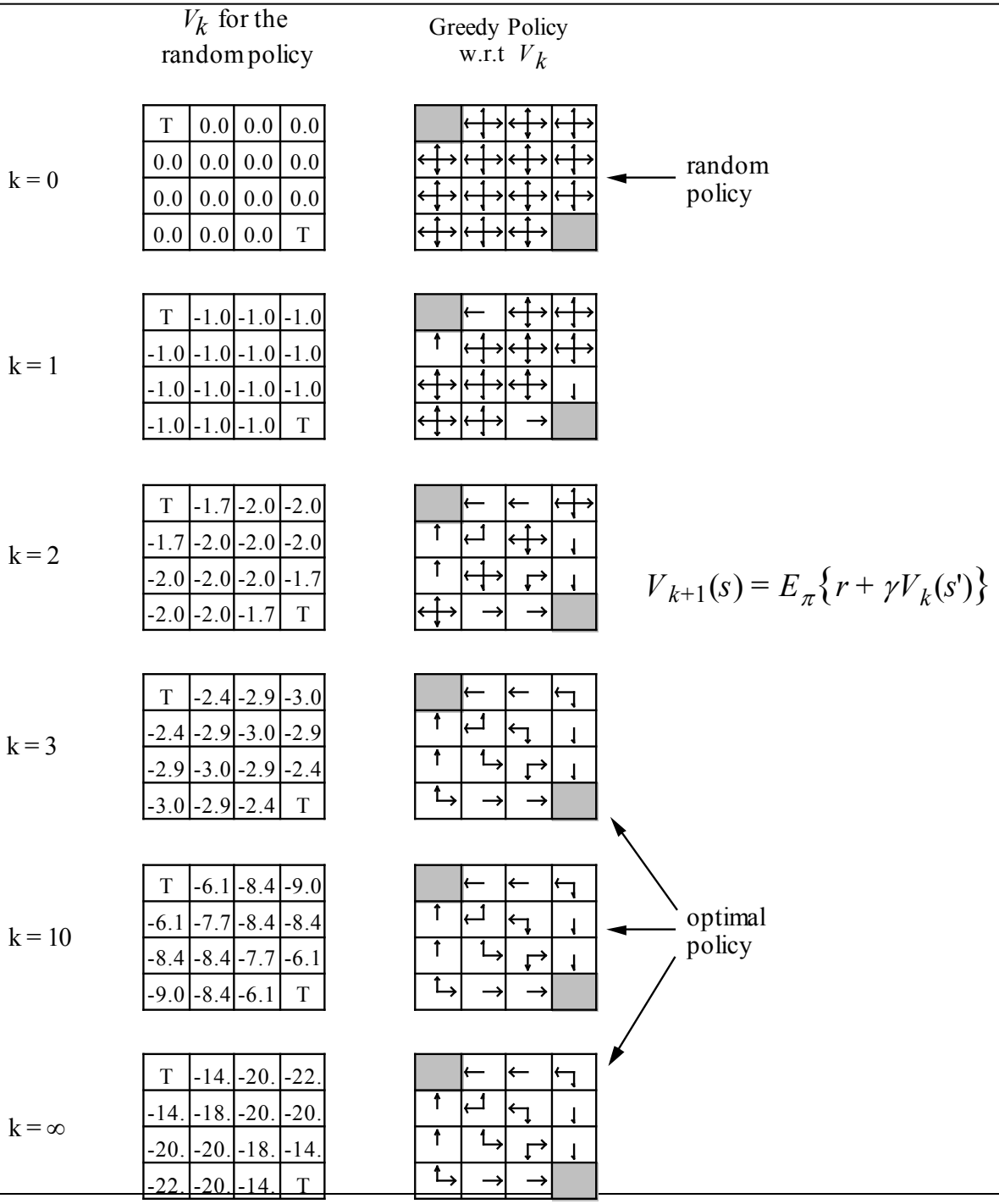


Optimal Policies

$\pi^*$

$Q^*$?

$Q^{\pi}$?


$V_k$ for the random policy
Greedy Policy w.r.t $V_k$
k = 0
T 0.0 0.0 0.0
0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0
0.0 0.0 0.0 T
random policy
k = 1
T -1.0 -1.0 -1.0
-1.0 -1.0 -1.0 -1.0
-1.0 -1.0 -1.0 -1.0
-1.0 -1.0 -1.0 T
k = 2
T -1.7 -2.0 -2.0
-1.7 -2.0 -2.0 -2.0
-2.0 -2.0 -2.0 -1.7
-2.0 -2.0 -1.7 T
$V_{k+1}(s) = E_\pi\{r + \gamma V_k(s')\}$
k = 3
T -2.4 -2.9 -3.0
-2.4 -2.9 -3.0 -2.9
-2.9 -3.0 -2.9 -2.4
-3.0 -2.9 -2.4 T
k = 10
T -6.1 -8.4 -9.0
-6.1 -7.7 -8.4 -8.4
-8.4 -8.4 -7.7 -6.1
-9.0 -8.4 -6.1 T
optimal policy
k = ∞
T -14. -20. -22.
-14. -18. -20. -20.
-20. -20. -18. -14.
-22. -20. -14. T

# Podsumowanie: optymalna strategia

$$\pi \geq \pi' \iff V^{\pi}(s) \geq V^{\pi'}(s), \forall s \in S$$

Strategia optymalna to każda taka, która jest większa równa od dowolnej innej. Oznaczać ją będziemy jako $\Pi^*$, zaś odpowiadające jej wartości stanu i akcji przez $V^*$ i $Q^*$. Zachodzi, że :

$$V^*(s) = \max_{\pi} V^{\pi}(s)$$

$$Q^*(s,a) = E\left\{r_{t+1} + \gamma V^*(s_{t+1}) \mid s_t = s, a_t = a\right\}$$

$$Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a)$$

dla każdych $s \in S$, $a \in A$.

# Wnioski:

Ponieważ z $V^*$ lub $Q^*$ łatwo odczytać $\Pi^*$, więc znalezienie optymalnej strategii sprowadza się do rozwiązania układu równań nieliniowych.
W praktyce bywa jednak to niemożliwe ze względu na:

- nieznajomość dynamiki MDP
- zbyt dużą liczbę stanów
- niespełnialność kryterium Markowa.

Wiele z pokazanych dalej algorytmów ma na celu realizowanie całej idei nawet gdy występują powyższe problemy.

# Programowanie dynamiczne (**DP**)

Algorytmy tak określane będą :

- wymagały znajomości dynamiki MDP
- trudne obliczeniowo
- występuje u nich bootstrapping
- ważne teoretycznie.

# Wartościowanie strategii dla DP

- Wyznacznie funkcji wartości stanu dla danej strategii, wedle iteracyjnego wzoru:

$$V_{k+1}(s) \leftarrow \max_{a \in A} \sum_{s' \in S} p^a_{ss'} \left( r^a_{ss'} + \gamma V_k(s') \right)$$

- Gwarantowana zbieżność do szukanej funkcji
- Pokazany na następnej stronie algorytm jest wersją jednotablicową, dla której szybciej obserwuje się zbieżność.

Input $\pi$, the policy to be evaluated

Initialize $V(s) = 0$, for all $s \in \mathcal{S}^+$

Repeat

$\Delta \leftarrow 0$

For each $s \in \mathcal{S}$:

$v \leftarrow V(s)$

$V(s) \leftarrow \sum_a \pi(s,a) \sum_{s'} \mathcal{P}^a_{ss'} \left[ \mathcal{R}^a_{ss'} + \gamma V(s') \right]$

$\Delta \leftarrow \max(\Delta, |v - V(s)|)$

until $\Delta < \theta$ (a small positive number)

Output $V \approx V^\pi$

# Polepszanie strategii dla DP

$\Pi, \Pi'$ - strategie deterministyczne. Wówczas, jeśli dla każdego $s \in S$ :

(1) $$Q^{\pi}(s, \pi'(s)) \geq V^{\pi}(s).$$

to dla każdego $s \in S$ :

(2) $$V^{\pi'}(s) \geq V^{\pi}(s)$$

zaś jeśli dla jakiegoś s jest ostra nierówność w (1) , to i w (2).
Zatem biorąc dla $\Pi$ strategię zachłanną $\Pi'$ spełniającą

$$\pi'(s) = \arg\max_a Q^{\pi}(s,a) = \arg\max_a \sum_{s'} \mathcal{P}^a_{ss'} \left[ \mathcal{R}^a_{ss'} + \gamma V^{\pi}(s') \right],$$

dostajemy $\Pi' \geq \Pi$, a jeśli $\Pi' = \Pi$, to obie te strategie są optymalne, gdyż spełniona jest równość optymalności Bellmana:

$$\begin{aligned} V^{\pi'}(s) &= \max_a E\left\{ r_{t+1} + \gamma V^{\pi'}(s_{t+1}) \mid s_t = s, a_t = a \right\} \\ &= \max_a \sum_{s'} \mathcal{P}^a_{ss'} \left[ \mathcal{R}^a_{ss'} + \gamma V^{\pi'}(s') \right]. \end{aligned}$$

# Algorytm iteracyjnego polepszania strategii dla DP

$$\pi_0 \xrightarrow{\text{E}} V^{\pi_0} \xrightarrow{\text{I}} \pi_1 \xrightarrow{\text{E}} V^{\pi_1} \xrightarrow{\text{I}} \pi_2 \xrightarrow{\text{E}} \cdots \xrightarrow{\text{I}} \pi^* \xrightarrow{\text{E}} V^*$$

Na zmianę wartościujemy strategię i polepszamy, wedlug schematu:

1. Initialization
   $V(s) \in \Re$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Repeat
   $\Delta \leftarrow 0$
   For each $s \in \mathcal{S}$:
   $v \leftarrow V(s)$
   $V(s) \leftarrow \sum_{s'} \mathcal{P}_{ss'}^{\pi(s)} \left[ \mathcal{R}_{ss'}^{\pi(s)} + \gamma V(s') \right]$
   $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number)

3. Policy Improvement
   *policy-stable* $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
   $b \leftarrow \pi(s)$
   $\pi(s) \leftarrow \arg\max_a \sum_{s'} \mathcal{P}_{ss'}^{a} \left[ \mathcal{R}_{ss'}^{a} + \gamma V(s') \right]$
   If $b \neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop; else go to 2

Zalety algorytmu:

- gwarantowane dojście do rozwiązania optymalnego w skończonej liczbie kroków
- wartościowanie strategii liczone tylko w pierwszej pętli od początku. dzięki czemu algorytm zbiega często zaskakująco szybko.

# Algorytm iteracyjnego wartościowania dla DP

W kolejnych pętlach modyfikujemy jedynie funkcję wartości stanu zgodnie z regułą:

$$V_{k+1}(s) = \max_a \sum_{s'} \mathcal{P}^a_{ss'} \left[ \mathcal{R}^a_{ss'} + \gamma V_k(s') \right],$$

Initialize $V$ arbitrarily, e.g., $V(s) = 0$, for all $s \in \mathcal{S}^+$

Repeat
  $\Delta \leftarrow 0$
  For each $s \in \mathcal{S}$:
    $v \leftarrow V(s)$
    $V(s) \leftarrow \max_a \sum_{s'} \mathcal{P}^a_{ss'} \left[ \mathcal{R}^a_{ss'} + \gamma V(s') \right]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)

Output a deterministic policy, $\pi$, such that
  $\pi(s) = \arg\max_a \sum_{s'} \mathcal{P}^a_{ss'} \left[ \mathcal{R}^a_{ss'} + \gamma V(s') \right]$

Dwa spojrzenia na algorytm:

- jako wersja poprzedniego z jedną pętlą wartościowania
- jako dążenie do równości optymalności Bellmana.

Kolejny algorytm zbiegający do $V^*$, a więc i $\Pi^*$.

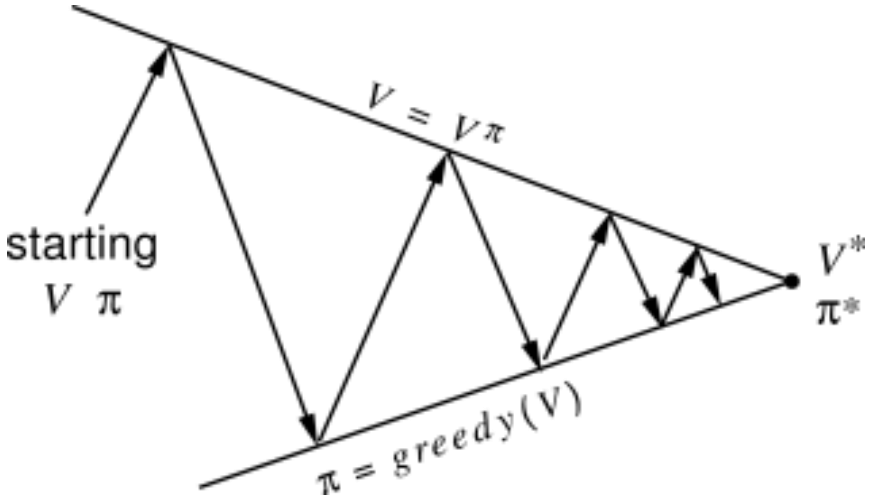
Naturalnym rozszerzeniem algorytmu jest wersja z kilkukrotną pętlą wartościowania – wtedy często szybsza zbieżność.

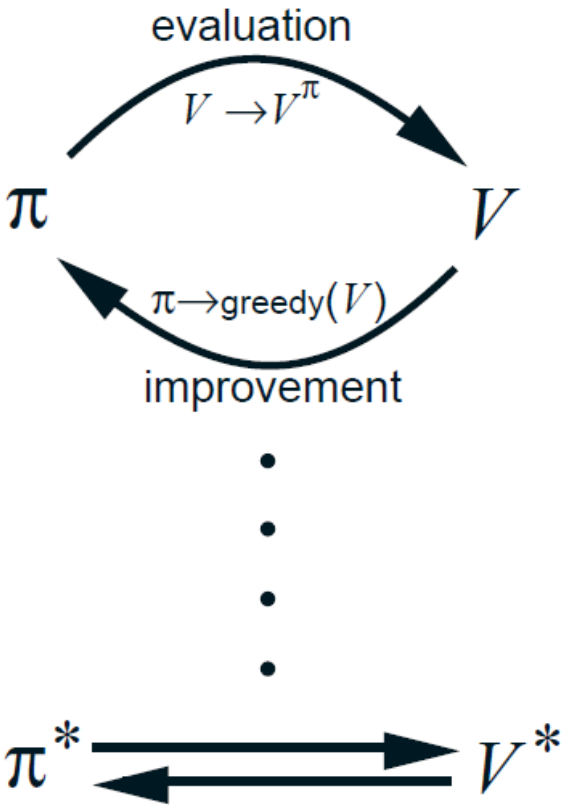
# Przyspieszenia DP

Algorytmy DP nie nadają się dla zbyt dużej ilości stanów.

Możliwe sposoby radzenia sobie z tym problemem to:

- asynchroniczne DP
- ogólniej GPI ( Generalized Policy Iteration):



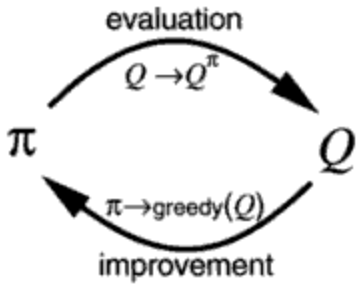



# Metody Monte Carlo

- Nie wymagają znajomości dynamiki MDP
- Idea jak w DP, potrzebne wartości zdobywają na przykładach ( uśrednianie Return )
- Wymagają zadania epizodycznego
- Uaktualnianie z epizodu na epizod.

$$Q^{\pi_k}\left(s, \pi_{k+1}(s)\right) = Q^{\pi_k}(s, \arg\max_a Q^{\pi_k}(s,a))$$

$$= \max_a Q^{\pi_k}(s,a)$$




$$\pi(s) = \arg\max_a Q(s,a).$$

# Wartościowanie strategii dla MC

Initialize:
$\pi \leftarrow$ policy to be evaluated
$V \leftarrow$ an arbitrary state-value function
$Returns(s) \leftarrow$ an empty list, for all $s \in \mathcal{S}$

Repeat forever:
(a) Generate an episode using $\pi$
(b) For each state $s$ appearing in the episode:
$R \leftarrow$ return following the first occurrence of $s$
Append $R$ to $Returns(s)$
$V(s) \leftarrow$ average($Returns(s)$)

Algorytm polega na zwykłym uśrednianiu Return, co dla dużej ilości przykładów przybliża wartość oczekiwaną.

Są dwa rodzaje takich algorytmów:

- every – visit MC
- first – visit MC ( ten po lewej).

Obie wersje zbiegają do funkcji wartości strategii *Π*.
Widzimy, że w przeciwieństwie do DP nie ma bootstrappingu.
Zaletą MC jest fakt, iż złożoność obliczeniowa nie zależy od liczby stanów ( uaktualnianie *V* tylko dla tych stanów, którymi zainteresowana *Π)*.

# Warościowanie akcji przy danej strategii dla MC

- rzecz potrzebna do algorytmów poprawy strategii
- algorytm analogiczny jak dla wartościowania strategii
- problemem jest jednak konflikt eksploracja – eksploatacja
- dwa podejścia do tego problemu to *exploring starts* i stochastyczne strategie.

# Uczenie się strategii dla MC (wersja podstawowa)

Działanie według schematu : $\pi_0 \xrightarrow{\text{E}} Q^{\pi_0} \xrightarrow{\text{I}} \pi_1 \xrightarrow{\text{E}} Q^{\pi_1} \xrightarrow{\text{I}} \pi_2 \xrightarrow{\text{E}} \cdots \xrightarrow{\text{I}} \pi^* \xrightarrow{\text{E}} Q^*$
czyli na zmianę dopasowywanie $Q$ do strategii i wzięcie strategii zachłannej względem tej Q.

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:
$Q(s,a) \leftarrow$ arbitrary
$\pi(s) \leftarrow$ arbitrary
$Returns(s,a) \leftarrow$ empty list

Repeat forever:
(a) Generate an episode using exploring starts and $\pi$
(b) For each pair $s,a$ appearing in the episode:
$R \leftarrow$ return following the first occurrence of $s,a$
Append $R$ to $Returns(s,a)$
$Q(s,a) \leftarrow \text{average}(Returns(s,a))$
(c) For each $s$ in the episode:
$\pi(s) \leftarrow \arg\max_a Q(s,a)$

- konieczna jest tu idea GPI
- przy założeniu eksploracyjnego startu i nieskończonej ilości prób – jest zbieżność do $\Pi^*$
- algorytm unieruchamia się dopiero gdy osiągnięto $\Pi^*$

# Uczenie się strategii dla MC (wersja on – policy)

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:
$Q(s,a) \leftarrow$ arbitrary
$Returns(s,a) \leftarrow$ empty list
$\pi \leftarrow$ an arbitrary $\varepsilon$-soft policy

Repeat forever:
(a) Generate an episode using $\pi$
(b) For each pair $s,a$ appearing in the episode:
$R \leftarrow$ return following the first occurrence of $s,a$
Append $R$ to $Returns(s,a)$
$Q(s,a) \leftarrow \text{average}(Returns(s,a))$
(c) For each $s$ in the episode:
$a^* \leftarrow \arg\max_a Q(s,a)$
For all $a \in \mathcal{A}(s)$:

$$\pi(s,a) \leftarrow \begin{cases} 1-\varepsilon+\varepsilon/|\mathcal{A}(s)| & \text{if } a = a^* \\ \varepsilon/|\mathcal{A}(s)| & \text{if } a \neq a^* \end{cases}$$

- analogiczny algorytm, używamy tylko strategii ε – *soft*, *greedy*, zamiast *exploring starts*
- przy polepszaniu strategii bierzemy ε – zachłanną wersję dotychczasowej strategii ( opisanej przez *Q)*
- można pokazać, że $\Pi$ - wersja ε - zachłanna strategii $\Pi'$ spełnia $V^{\pi}(s) \geq V^{\pi'}(s)$

dla każdego s ∈ S, zaś stabilność osiągana, gdy $\Pi$, $\Pi'$ będą optymalne wśród wszystkich ε - *soft* strategii.

Wadą algorytmu jest fakt, iż nauczona strategia jest ε - zachłanna, a więc nieoptymalna.

# Uczenie się strategii dla MC (wersja off – policy)

Uczymy się tutaj optymalnych wartości strategii, używając do eksploracji niezależnej od nich innej strategii.
Załóżmy, że w kolejnych epizodach używamy niezmiennej strategii *Π'*, zaś chcemy nauczyć się strategii *Π*. Niech $p_i(s)$, $p_i'(s)$ odnoszą się do i – tej wizyty stanu s i oznaczają prawdopodobieństwa zobaczenia sekwencji zdarzeń takiej jaka nastąpiła odpowiednio dla strategii *Π* i *Π'*, zaś $R_i(s)$ – zaobserwowany wtedy Return. Wówczas oszacowanie na wartość *V(s)* dla strategii *Π* wynosi:

$$V(s) = \frac{\sum_{i=1}^{n_s} \frac{p_i(s)}{p_i'(s)} R_i(s)}{\sum_{i=1}^{n_s} \frac{p_i(s)}{p_i'(s)}}.$$

gdzie : $p_i(s_t) = \prod_{k=t}^{T_i(s)-1} \pi(s_k, a_k) \mathcal{P}_{s_k s_{k+1}}^{a_k}$

i szczęśliwie: $\frac{p_i(s_t)}{p_i'(s_t)} = \frac{\prod_{k=t}^{T_i(s)-1} \pi(s_k, a_k) \mathcal{P}_{s_k s_{k+1}}^{a_k}}{\prod_{k=t}^{T_i(s)-1} \pi'(s_k, a_k) \mathcal{P}_{s_k s_{k+1}}^{a_k}} = \prod_{k=t}^{T_i(s)-1} \frac{\pi(s_k, a_k)}{\pi'(s_k, a_k)}.$

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:

$Q(s,a) \leftarrow$ arbitrary

$N(s,a) \leftarrow 0$ ; Numerator and

$D(s,a) \leftarrow 0$ ; Denominator of $Q(s,a)$

$\pi \leftarrow$ an arbitrary deterministic policy

Repeat forever:

(a) Select a policy $\pi'$ and use it to generate an episode:

$s_0, a_0, r_1, s_1, a_1, r_2, \ldots, s_{T-1}, a_{T-1}, r_T, s_T$

(b) $\tau \leftarrow$ latest time at which $a_\tau \neq \pi(s_\tau)$

(c) For each pair $s, a$ appearing in the episode after $\tau$:

$t \leftarrow$ the time of first occurrence (after $\tau$) of $s, a$

$w \leftarrow \prod_{k=t+1}^{T-1} \frac{1}{\pi'(s_k, a_k)}$

$N(s,a) \leftarrow N(s,a) + wR_t$

$D(s,a) \leftarrow D(s,a) + w$

$Q(s,a) \leftarrow \frac{N(s,a)}{D(s,a)}$

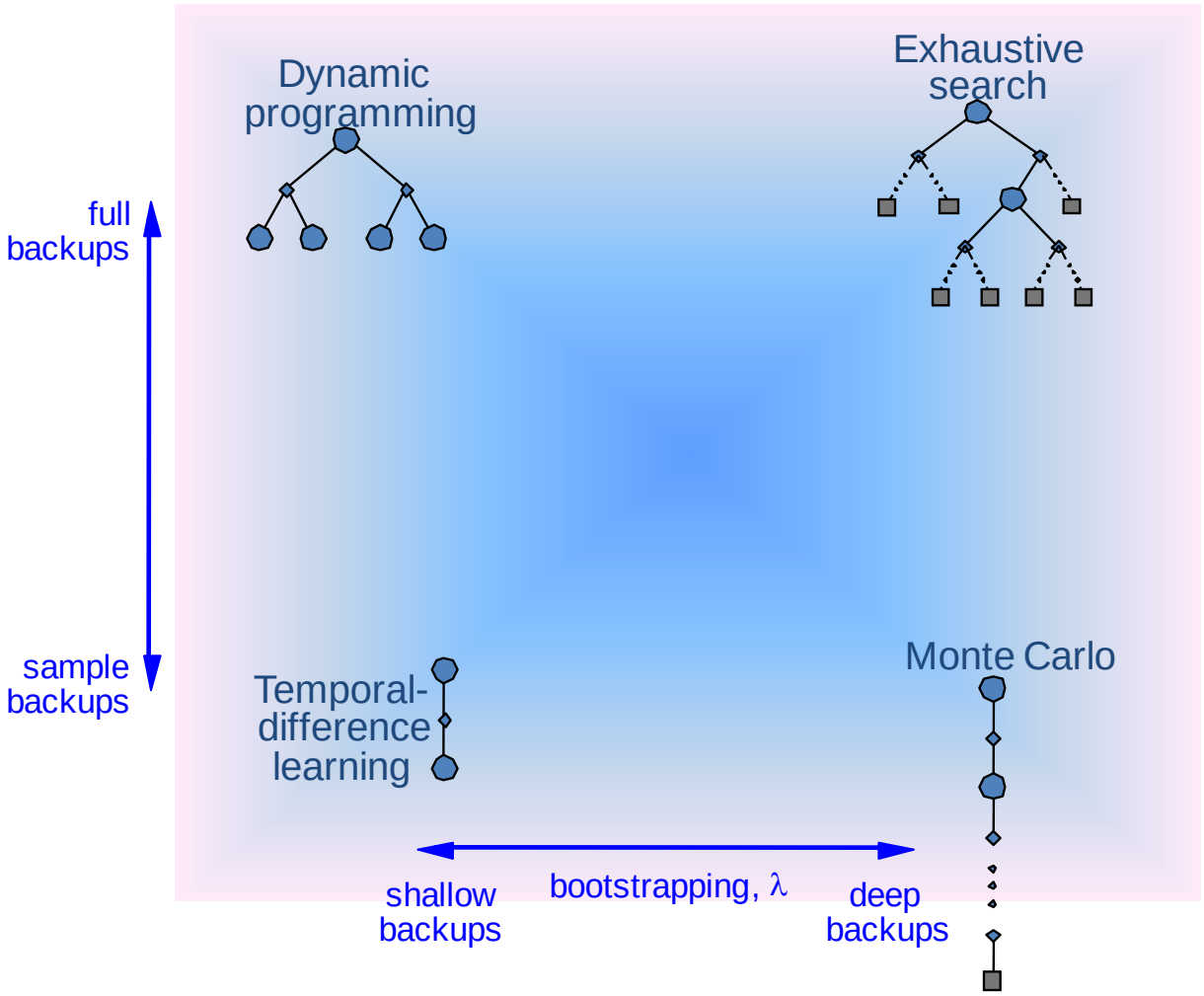
(d) For each $s \in \mathcal{S}$:

$\pi(s) \leftarrow \arg\max_a Q(s,a)$

Zaleta algorytmu : możemy uczyć się strategii zachłannej ( zmierzając w stronę optymalności, zaś do eksploracji możemy użyć dopuszczającej każdą parę (stan – akcja) strategii.

Wada : powolność algorytmu.

# Typy pamięci zapasowej



# Temporal – Difference Learning

- Przesiąknięte pomysłami z DP i MC.
- Potrzebne wartości wyciągane na bazie przykładów ( nie potrzebna znajomość dynamiki ) – podobnie jak w MC.
- Bootstrapping – podobnie jak w DP.

# Wartościowanie strategii dla TD

Algorytm *constant - $\alpha$ MC* dla wartościowania strategii polega na aktualizacjach :

$$V(s_t) \leftarrow V(s_t) + \alpha\Big[R_t - V(s_t)\Big],$$

Zaś w TD pomysł na aktualizację ( już nie co epizod, a co chwilę), jest taki :

$$V(s_t) \leftarrow V(s_t) + \alpha\Big[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)\Big].$$

Zatem przy użyciu bootstrappingu (jak w DP) szacujemy wartość $R_t$ – jednak całe to oszacowanie bierze się z pobrania napatkanych w doświadczeniu wartości – zatem jak w MC.

Initialize $V(s)$ arbitrarily, $\pi$ to the policy to be evaluated
Repeat (for each episode):
  Initialize $s$
  Repeat (for each step of episode):
    $a \leftarrow$ action given by $\pi$ for $s$
    Take action $a$; observe reward, $r$, and next state, $s'$
    $V(s) \leftarrow V(s) + \alpha\big[r + \gamma V(s') - V(s)\big]$
    $s \leftarrow s'$
  until $s$ is terminal

# Zalety TD :

• przewaga TD nad DP - nie trzeba znać dynamiki środowiska

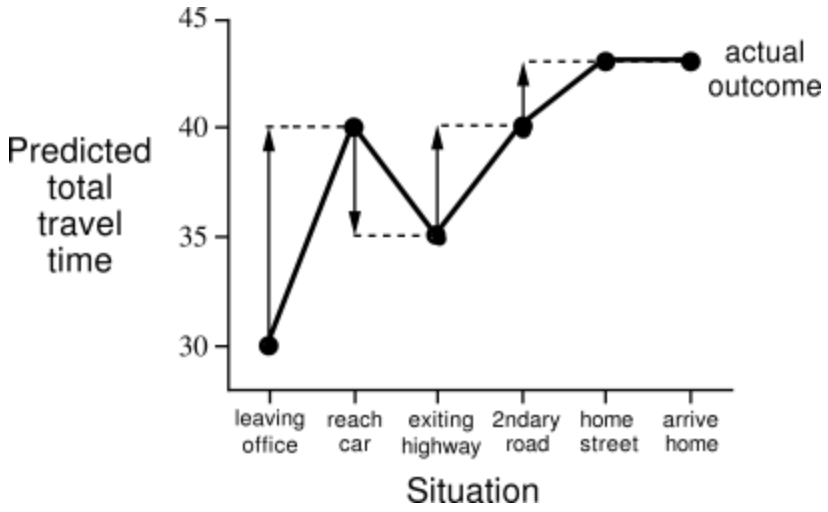
• przewaga TD nad MC – w MC aktualizacja po epizodzie (gdy epizod jest bardzo długi, albo w ogóle ich nie ma?),

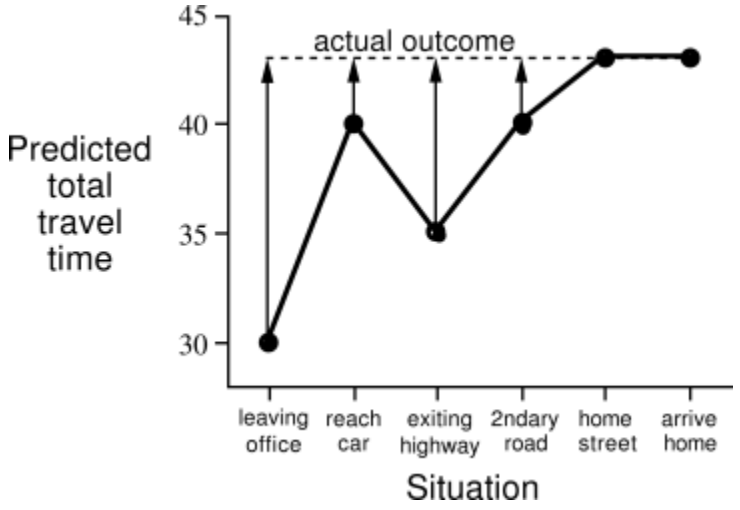
• w TD – aktualizacja na bieżąco – co poza innymi zaletami umożliwia rozpatrywanie zadań ciągłych

• przy wszystkim tym algorytm TD gwarantuje również zbieżność do szukanej funkcji, pod warunkiem odpowiedniego dobierania kroku $\alpha$.

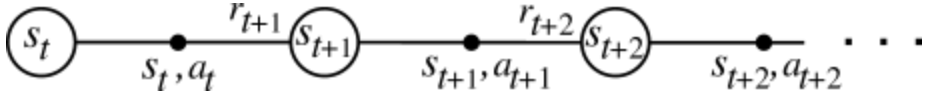
• algorytm TD ma tendencję do szybszej zbieżności niż MC.

# TD vs MC


45
40
35
30
Predicted total travel time
actual outcome
leaving office
reach car
exiting highway
2ndary road
home street
arrive home
Situation


45
40
35
30
Predicted total travel time
actual outcome
leaving office
reach car
exiting highway
2ndary road
home street
arrive home
Situation

# Sarsa
## (uczenie się strategii dla TD, on-policy)




- znów idea GPI
- znów konflikt eksploracja – eksploatacja
- znów potrzebne nam przybliżenie $Q$ dla danej strategii, uzyskujemy je tak jak $V$ :

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\left[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)\right].$$

Initialize $Q(s, a)$ arbitrarily
Repeat (for each episode):
  Initialize $s$
  Choose $a$ from $s$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
  Repeat (for each step of episode):
    Take action $a$, observe $r$, $s'$
    Choose $a'$ from $s'$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
    $Q(s, a) \leftarrow Q(s, a) + \alpha\left[r + \gamma Q(s', a') - Q(s, a)\right]$
    $s \leftarrow s'$; $a \leftarrow a'$;
  until $s$ is terminal

Zachodzi zbieżność do $\Pi^*$ pod warunkiem strategii $\varepsilon$ $-$ *soft, greedy* z $\varepsilon \rightarrow 0$.

# Q – learning

## (uczenie się strategii dla TD, off – policy)

- uaktualnianie użyte do przybliżania $Q$ jest tu postaci

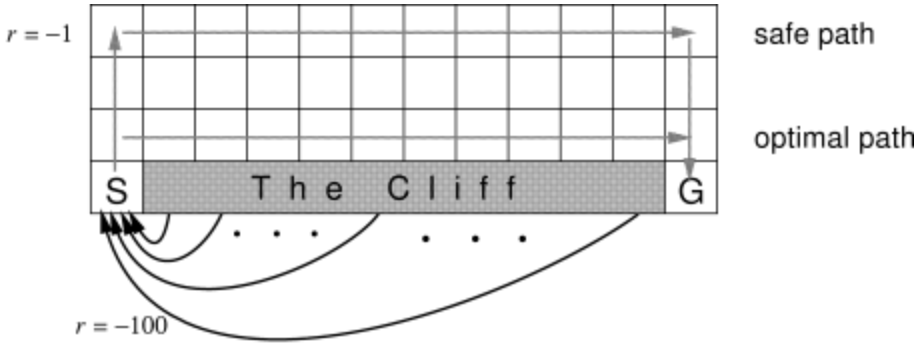
$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)\right].$$

- widać wyraźne nawiązanie do równania optymalności Bellmana
- zaleta wzoru jest taka, iż możemy wybrać dowolną stałą strategię, np. dobrze eksplorującą, niezależnie ucząc się $Q$

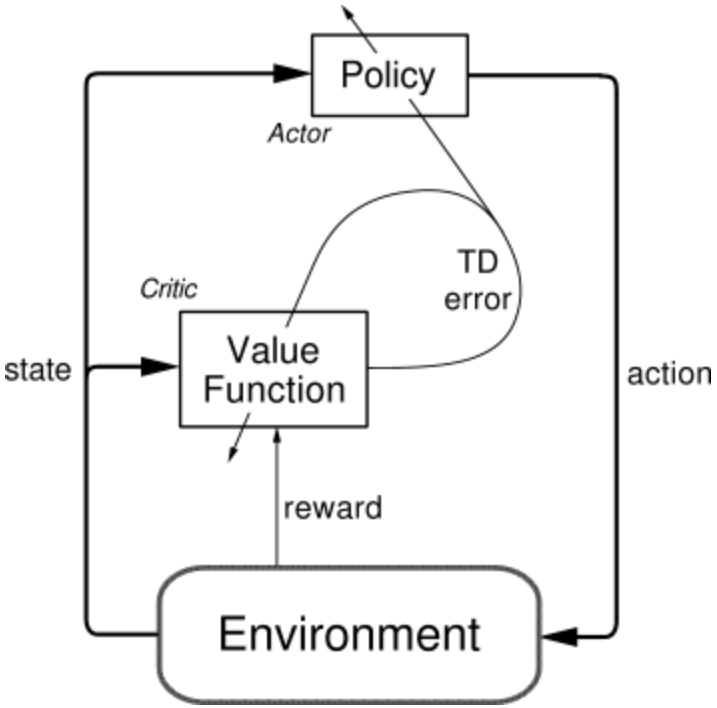
Initialize $Q(s, a)$ arbitrarily
Repeat (for each episode):
  Initialize $s$
  Repeat (for each step of episode):
    Choose $a$ from $s$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
    Take action $a$, observe $r$, $s'$
    $Q(s, a) \leftarrow Q(s, a) + \alpha\left[r + \gamma \max_{a'} Q(s', a') - Q(s, a)\right]$
    $s \leftarrow s'$;
  until $s$ is terminal

Przy założeniu strategii eksplorującej i odpowiednim zachowaniu $\alpha$ algorytm zbiega do $Q^*$ i to często bardzo szybko.

# Sarsa vs Q - learning




# Metody aktor – krytyk



# Ślady aktywności

• zobaczymy teraz uogólnienie dwóch spośród poznanych do tej pory metot : MC i TD

• uogólnienie to pozwala osiągnąć lepsze rezultaty ( czyli przede wszystkim szybszą zbieżność ) niż te dwie skrajne metody

• algorytmy te określimy mianem TD($\lambda$) gdzie $\lambda$ -parametr z przedziału [0, 1]

• zobaczymy dwa spojrzenia na ten algorytm: do przodu (teoretyczne) i do tyłu (mechaniczne).

# n – krokowy algorytm TD dla wartościowania strategii

W algorytmie TD mamy 1 – krokową aktualizację: $R_t^{(1)} = r_{t+1} + \gamma V_t(s_{t+1})$.

W algorytmie MC mamy pełną aktualizcję: $R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots + \gamma^{T-t-1} r_T$,

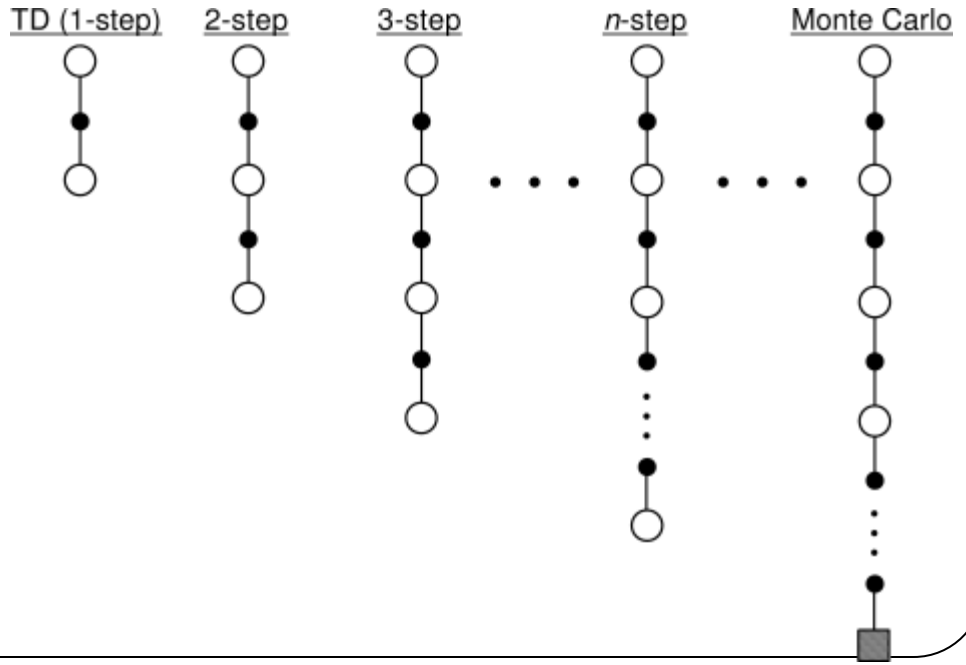
Moglibyśmy zatem pomyśleć n – krokowej aktualizacji, gdzie n – ustalone:

$$R_t^{(n)} = r_{t+1} + \gamma r_{t+2} + \gamma^2 + \cdots + \gamma^{n-1} r_{t+n} + \gamma^n V_t(s_{t+n}).$$

Idea TD używając do aktualizacji takiego Return wciąż będzie aktualna.

Dla n – krokowej metody przyjmujemy, że dla $T - t < n$ zachodzi $R_t^{(n)} = R_t^{(T-t)} = R_t$

( albo nic nie zakładamy, jeśli pamiętamy o uogólnionym spojrzeniu na zadania epizodyczne i ciągłe ).




Analogicznie jak w MC i TD aktualizacja wartości stanu to:

$$\Delta V_t(s_t) = \alpha \left[ R_t^{(n)} - V_t(s_t) \right],$$

gdzie $\alpha$ - ustalony parametr.

# Metody on – line i off - line

Metody n – krokowe, a także inne poznane wcześniej / dalej dzielimy na:

• *on – line*, gdzie aktualizacja V dokonywana po każdym kroku

• *off – line*, gdzie V jest niezmienne przez cały epizod, zaś po jego zakończeniu następuje aktualizacja V do : $V(s) + \sum_{t=0}^{T-1} \Delta V_t(s)$

Dla obu metod zachodzi *własność redukcji błędu* :

$$\max_s \left| E_\pi \left\{ R_t^{(n)} \;\middle|\; s_t = s \right\} - V^\pi(s) \right| \;\leq\; \gamma^n \max_s |V(s) - V^\pi(s)| \,.$$

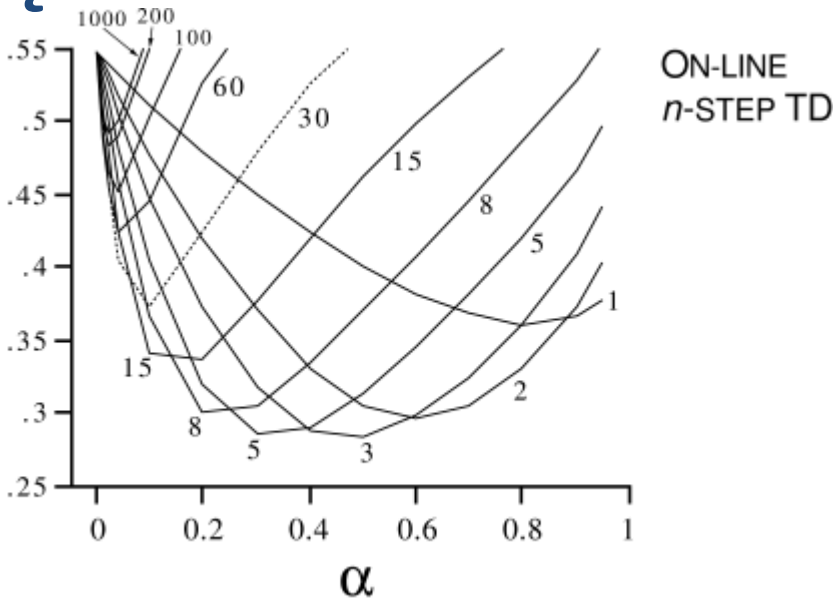
Można na jej bazie pokazać, że metody n – krokowe on / off - line zbiegają do $V^\pi$.

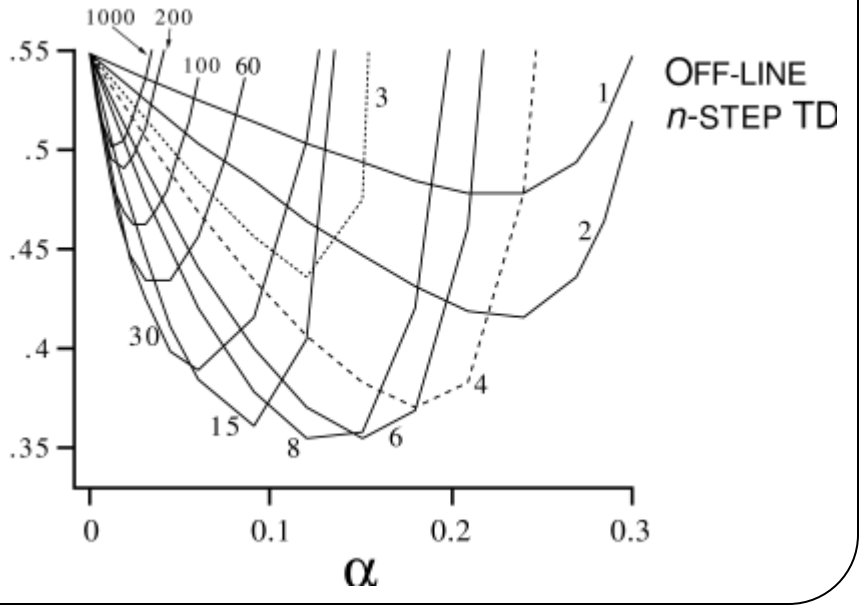
Wadą metod n – krokowych jest jednak fakt, iż nie są one wygodne do zaimplementowania.

Dalsze metody unikną tego problemu, zachowując całą ideę.

# Przykład – zadanie błądzenia

- dane jest 19 stanów: 1 .. 19
- możliwości ruchu: LEWO, PRAWO
- nagroda wyłącznie za dojście do stanu 19 – w wysokości 1
- stany końcowe to 1 i 19
- początkowa wartość każdego stanu to 0.5




Dla metody n – krokowej im większe n, tym szybciej dostrzegana nagroda. (tzn. w pierwszym epizodzie dalekie stany od stanu 19 są uaktualnione.

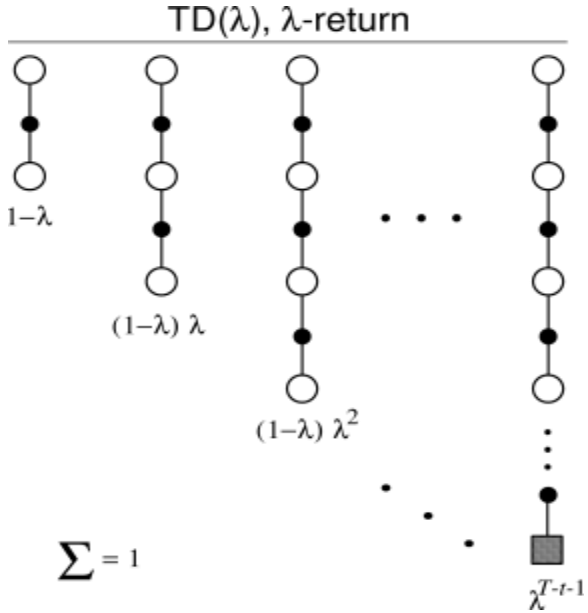
Po prawej mamy przedstawione średnie odległości znalezionych rezultatów od rezultatu docelowego ( wartości każdego stanu przy ustalonej strategii).

# Spojrzenie „do przodu” na TD(λ)

Pomysł jest taki, by tworzyć potrzebny do aktualizacji Return jako uśrednienie Return – ów n – krokowych dla różnych n, np. $R_t^{ave} = \frac{1}{2}R_t^{(2)} + \frac{1}{2}R_t^{(4)}$.
Algorytmy używające takich aktualizcją zachowają podstawową własność – zbieżność do $V^\pi$. Metoda TD(λ) „ do przodu” to szczególny sposób takiego uśredniania. Stosowane tu uśrednienie nazwiemy λ - return: $R_t^\lambda = (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1} R_t^{(n)}$.

Wzór ten ma sens, gdy pamiętamy o ujednoliceniu zadań epizodycznych i ciągłych.

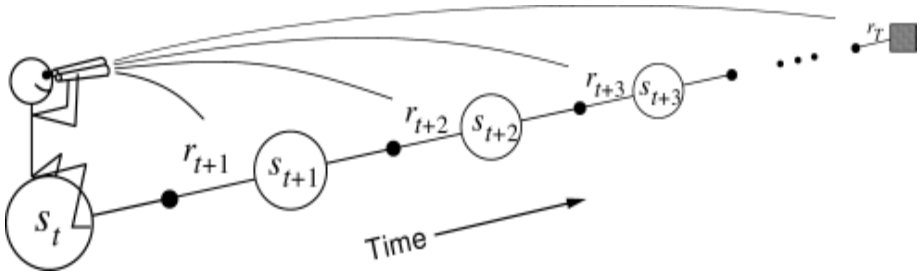
Możemy też wzór ten napisać tak: $R_t^\lambda = (1-\lambda)\sum_{n=1}^{T-t-1} \lambda^{n-1} R_t^{(n)} + \lambda^{T-t-1} R_t.$




Dla λ = 0 uaktualnienie przy użyciu λ - return – jak w metodzie TD ( którą teraz nazwiemy TD(0).

Dla λ = 1 uaktualnieneie jak w metodzie MC (TD(1)).

# Wartościowanie strategii dla TD(λ) „do przodu”




To tzw. $\lambda$ - *return algorithm*.

Aktualizacja dana jest wzorem:

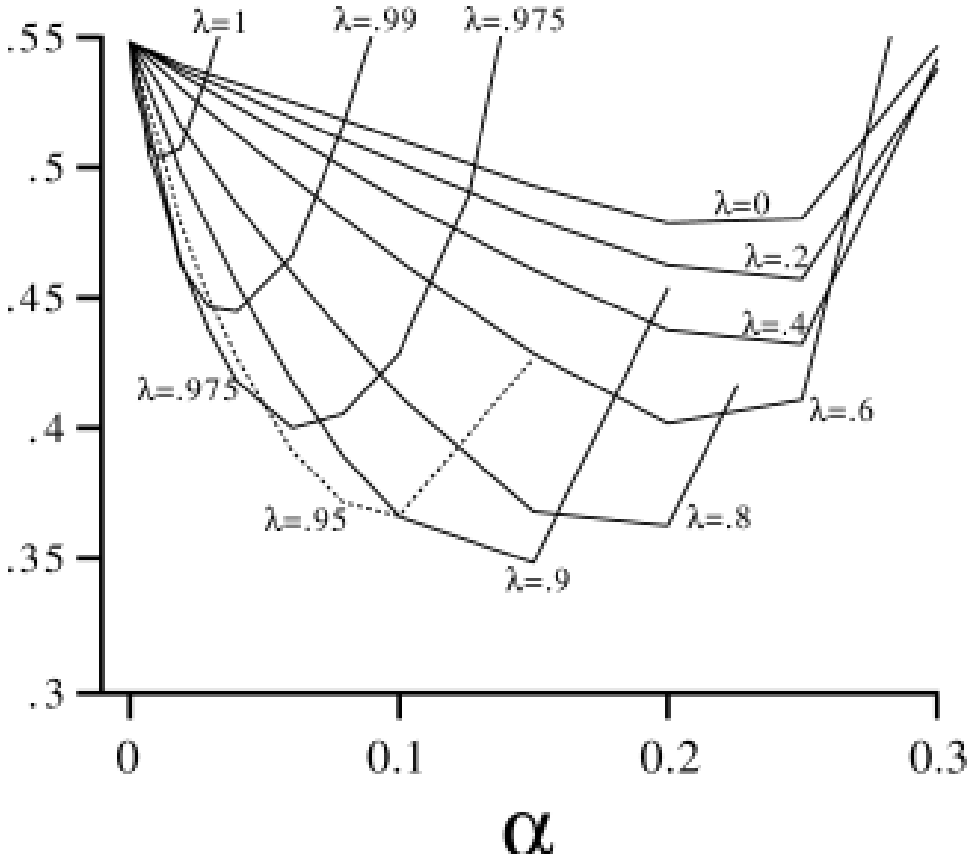
$$\Delta V_t(s_t) = \alpha\left[R_t^{\lambda} - V_t(s_t)\right].$$

Można ją stosować *off* bądź *on – line*.

Widzimy analogię algorytmu TD(λ) do metod n – krokowych, jako most między TD a MC.
Jednak nie można stwierdzić, że któraś z tych metod jest lepsza.
Przyczyną zainteresowania algorytmem $\lambda$ - *return* jest fakt, iż ma on prawie równoważny odpowiednik w postaci łatwego do zaimplementowania ( nie odwołującego się do przyszłości ) algorytmu, o którym będzie mowa za chwilę.

# Przykład działania *off-line*

$\lambda$ - *return algorithm* (dla zadania błądzenia)



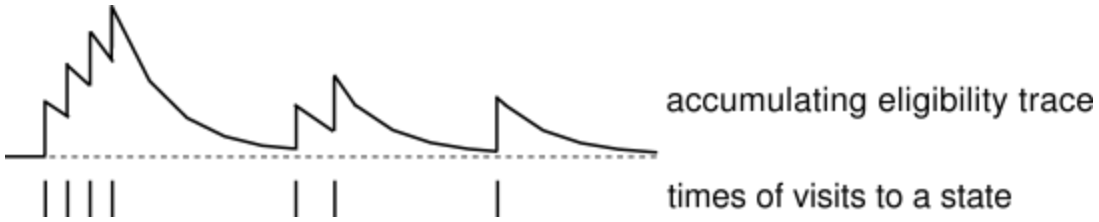
# Spojrzenie „do tyłu” na TD(λ)

Zobaczymy teraz algorytm łatwy do zaimplementowania, nie odwołujący się do przyszłości ( co pozwala strosować go na żywo ), który wbrew pozorom dla przypadku *off – line* jest dokładnie tym samym co $\lambda$ - *return algorithm*.

W algorytmie tym używane są tzw. *ślady aktywności,* które są modyfikowane przy odwiedzaniu kolejnych stanów w następujący sposób:

$$e_t(s) = \begin{cases} \gamma\lambda e_{t-1}(s) & \text{if } s \neq s_t; \\ \gamma\lambda e_{t-1}(s) + 1 & \text{if } s = s_t, \end{cases}$$

gdzie $s_t$ – to aktualny stan.
Ślad ten zwany jest akumulującym się śladem, co ilustruje taki przykład:

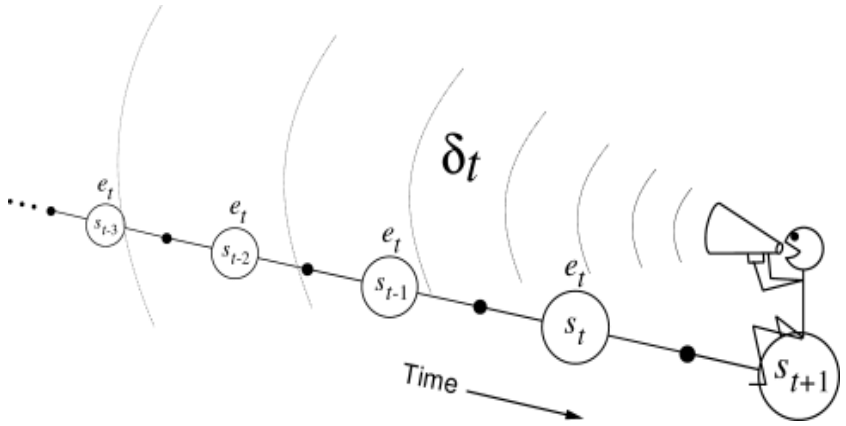



Błąd służący do aktualizcji jak w TD(0), tzn. $\delta_t = r_{t+1} + \gamma V_t(s_{t+1}) - V_t(s_t).$
Aktualizacja zaś dokonywana jest w każdym kroku dla każdego stanu:

$$\Delta V_t(s) = \alpha \delta_t e_t(s), \qquad \text{for all } s \in \mathcal{S}.$$

Znów aktualizacja może być dokonywana *off* bądź *on – line*.

# Wartościowanie strategii dla TD(λ) „do tyłu”




Initialize $V(s)$ arbitrarily and $e(s) = 0$, for all $s \in \mathcal{S}$
Repeat (for each episode):
  Initialize $s$
  Repeat (for each step of episode):
    $a \leftarrow$ action given by $\pi$ for $s$
    Take action $a$, observe reward, $r$, and next state, $s'$
    $\delta \leftarrow r + \gamma V(s') - V(s)$
    $e(s) \leftarrow e(s) + 1$
    For all $s$:
      $V(s) \leftarrow V(s) + \alpha \delta e(s)$
      $e(s) \leftarrow \gamma \lambda e(s)$
    $s \leftarrow s'$
  until $s$ is terminal

Przedstawiony algorytm jest wersją *on – line*.
Zauważmy, że dla λ = 0 metoda ta jest równoważna TD ( w wersji *off/on – line* ), zaś dla λ = 1 w wersji *off – line* równoważna every – visit – MC.
Ten algorytm jednak w przeciwieństwie do MC – jest możliwy do zastosowania w zadaniu nieepizodycznym. Poza tym metodę TD(λ) można przeprowadzać na żywo, co miewa duży wpływ np. na szybkość uczenia się strategii.

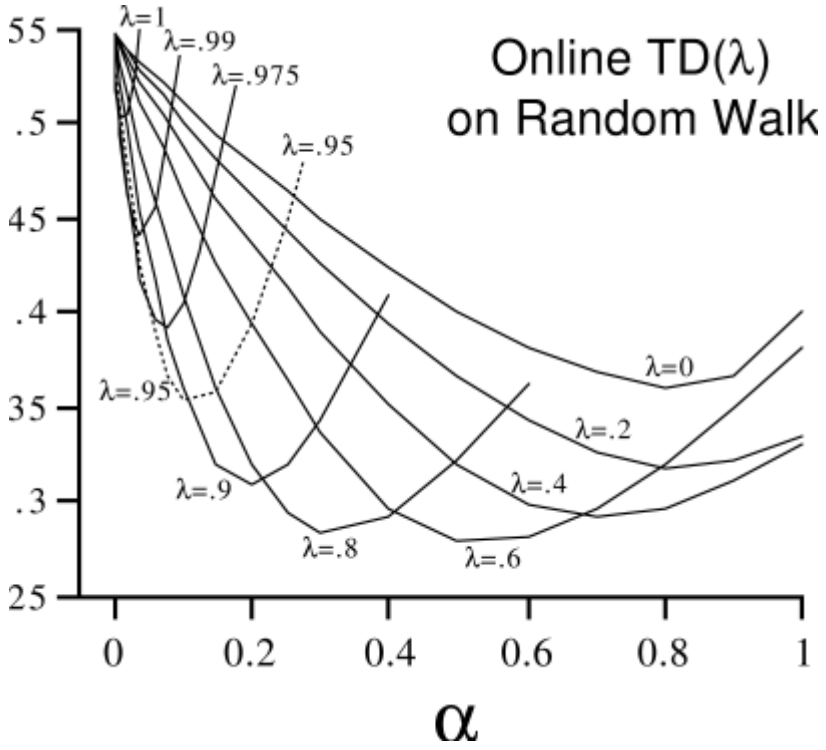
# Równoważność spojrzenia „do przodu” i „do tyłu”

Można łatwo pokazać, że metody TD(λ) „do przodu” i „do tyłu” w przypadku *off – line* dokonają po jednym epizodzie, startując z takich samych funkcji wartości stanu V na tym samym zadaniu, dokładnie takich samych uaktualnień tej funkcji.

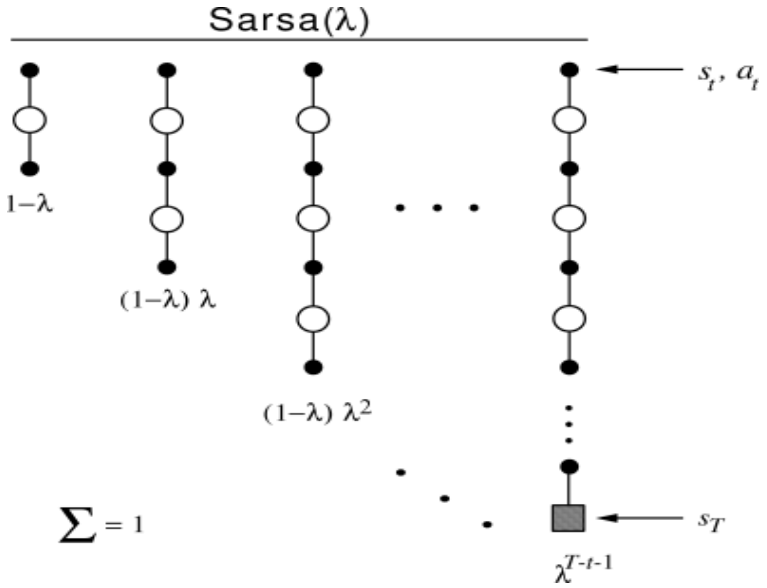
W przypadku algorytmów *on – line* różnica może być dowolnie mała, jeśli dobrać tylko odpowiednio mały krok $\alpha$ .

# Przykład działania *on – line* TD(λ)
# ( dla zadania błądzenia )

(Zauważmy, że działania *off – line* TD(λ) już widzieliśmy – bo to po prostu równoważne *λ - return algorithm)*



# Sarsa(λ)




Mamy tu wykorzystanie metody TD(λ) do nuki strategii – uogólnienie algorytmu SARSA ( który był przypadkiem Sarsa(0) ).
Metoda jest analogiczna do TD (λ) zastosowana tylko do funkcji Q. Po lewej pokazany jest schemat dla wersji algorytmu „do przodu", zaś znów użyjemy w praktyce wersji „do tyłu".

Każdorazowy błąd to tutaj :

$$\delta_t = r_{t+1} + \gamma Q_t(s_{t+1}, a_{t+1}) - Q_t(s_t, a_t)$$

Initialize $Q(s,a)$ arbitrarily and $e(s,a) = 0$, for all $s, a$
Repeat (for each episode):
  Initialize $s$, $a$
  Repeat (for each step of episode):
    Take action $a$, observe $r$, $s'$
    Choose $a'$ from $s'$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
    $\delta \leftarrow r + \gamma Q(s', a') - Q(s, a)$
    $e(s,a) \leftarrow e(s,a) + 1$
    For all $s, a$:
      $Q(s,a) \leftarrow Q(s,a) + \alpha \delta e(s,a)$
      $e(s,a) \leftarrow \gamma \lambda e(s,a)$
    $s \leftarrow s'$; $a \leftarrow a'$
  until $s$ is terminal

zaś uaktualnienie:

$$Q_{t+1}(s,a) = Q_t(s,a) + \alpha \delta_t e_t(s,a), \qquad \text{for all } s, a$$

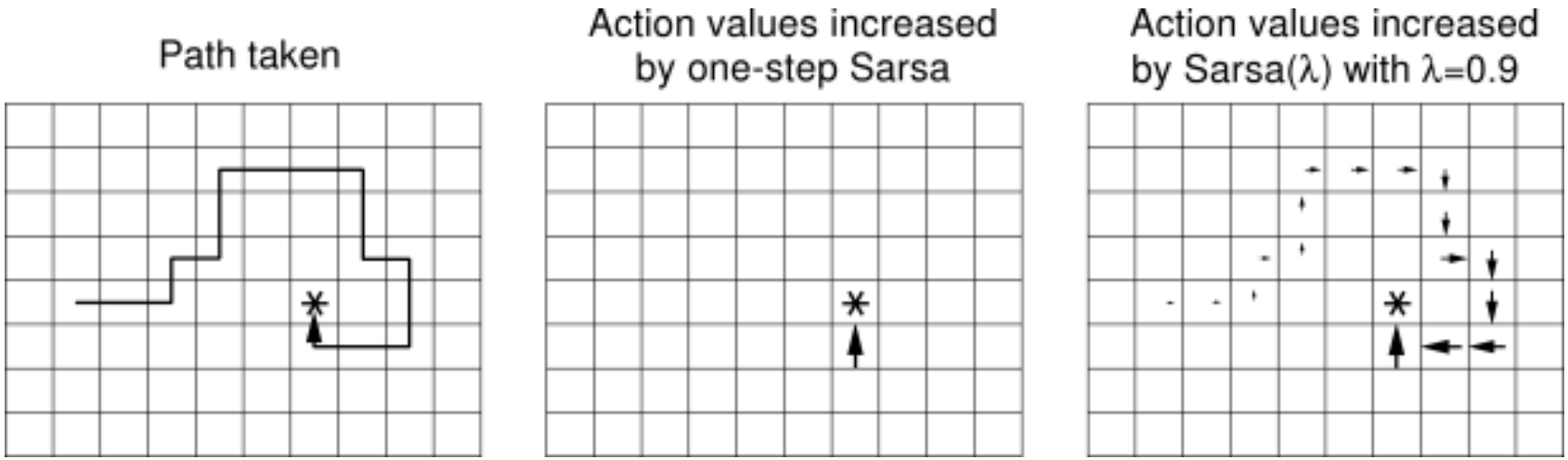
gdzie ślad aktywności trzymany dla każdej pary stan–akcja, uaktualniany:

$$e_t(s,a) = \begin{cases} \gamma \lambda e_{t-1}(s,a) + 1 & \text{if } s = s_t \text{ and } a = a_t; \\ \gamma \lambda e_{t-1}(s,a) & \text{otherwise.} \end{cases} \qquad \text{for all } s, a$$

Jest to algorytm *on – policy*, stosujący strategię ε – *greedy* – ale też niestety uczący się optymalnej wśród takich.

# Sarsa(λ) vs Sarsa(0)

( na przykładzie dwuwymiarowego zadania błądzenia, z jednym stanem końcowym *, jako jedynym dającym jakąś nagrodę )

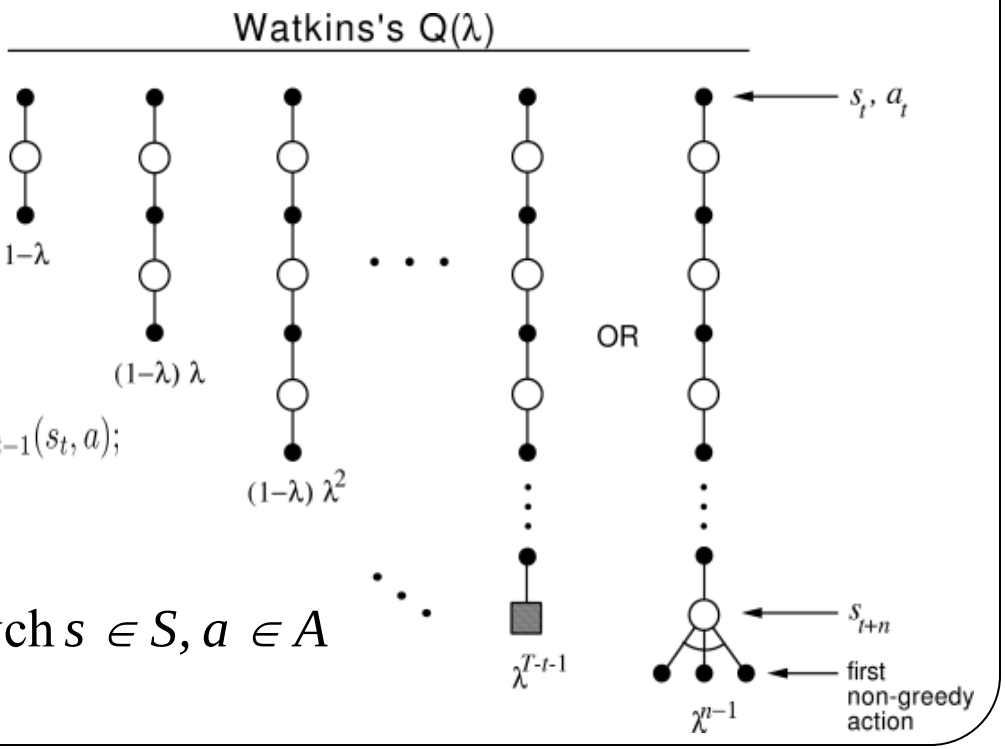


# Q(λ) ( metoda Watkins' a )

Jest to kolejne uogólnienie – tym razem metody Q – learning. Jest to algorytm *off – policy* : uczymy się strategii zachłannej, korzystając z eksplorującej wersji (*greedy*). Metoda wygląda podobnie do Sarsa, tylko jako że chcemy uczyć się strategii zachłannej ( nie ε – *greedy* jak Sarsa ) musimy uważać na to, by nie uczyć się na niezachłannych wyborach strategii eksplorującej. Zatem jeśli $a_{t+n}$ - to pierwsza eksplorująca akcja, najdłuższą składową $\lambda$ - *return* w metodzie „do przodu" będzie:

$$r_{t+1} + \gamma r_{t+2} + \cdots + \gamma^{n-1} r_{t+n} + \gamma^n \max_a Q_t(s_{t+n}, a)$$

Po prawej mamy schemat tworzenia tych Return – ów. Implementacja jest znów w wersji „do tyłu", tylko ograniczenie ogona przekłada się na czyszczenie śladów aktywności, gdy doszło do wyboru akcji niezachłannej:




$$e_t(s,a) = \mathcal{I}_{ss_t} \cdot \mathcal{I}_{aa_t} + \begin{cases} \gamma\lambda e_{t-1}(s,a) & \text{if } Q_{t-1}(s_t,a_t) = \max_a Q_{t-1}(s_t,a); \\ 0 & \text{otherwise,} \end{cases}$$

Uaktulnienie w każdym kroku to

$$Q_{t+1}(s,a) = Q_t(s,a) + \alpha\delta_t e_t(s,a), \quad \text{dla każdych } s \in S, a \in A$$

zaś $\delta_t = r_{t+1} + \gamma \max_{a'} Q_t(s_{t+1}, a') - Q_t(s_t, a_t).$

Initialize $Q(s,a)$ arbitrarily and $e(s,a) = 0$, for all $s, a$
Repeat (for each episode):
  Initialize $s$, $a$
  Repeat (for each step of episode):
    Take action $a$, observe $r$, $s'$
    Choose $a'$ from $s'$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
    $a^* \leftarrow \arg\max_b Q(s',b)$ (if $a'$ ties for the max, then $a^* \leftarrow a'$)
    $\delta \leftarrow r + \gamma Q(s',a^*) - Q(s,a)$
    $e(s,a) \leftarrow e(s,a) + 1$
    For all $s, a$:
      $Q(s,a) \leftarrow Q(s,a) + \alpha\delta e(s,a)$
      If $a' = a^*$, then $e(s,a) \leftarrow \gamma\lambda e(s,a)$
                    else $e(s,a) \leftarrow 0$
    $s \leftarrow s'$; $a \leftarrow a'$
  until $s$ is terminal

Wadą algorytmu jest to, że częste usuwanie dotychczasowych śladów sprawia, iż uczenie jest niewiele szybsze niż dla zwykłego Q – learning ( czyli Q(0) ).

Alternatywą są :

- algorytm Peng'a
- naiwna wersja algorytmu Watkinson'a.

# Uogólnienie metody aktor - krytyk

Rzecz jest analogiczna jak dla zwykłego algorytmu aktor - krytyk, tylko teraz :

• krytyk poprawia swoją funkcję wartości stanu przy użyciu TD (λ)

• dla metody bez śladów aktywności aktor poprawiał preferencje poszczególnych akcji zgodnie z regułą :

$$p_{t+1}(s,a) = \begin{cases} p_t(s,a) + \alpha\delta_t & \text{if } a = a_t \text{ and } s = s_t \\ p_t(s,a) & \text{otherwise,} \end{cases}$$

zaś tutaj uaktualnienie zachodzi dla każdej pary stan – akcja:

$$p_{t+1}(s,a) = p_t(s,a) + \alpha\delta_t e_t(s,a),$$

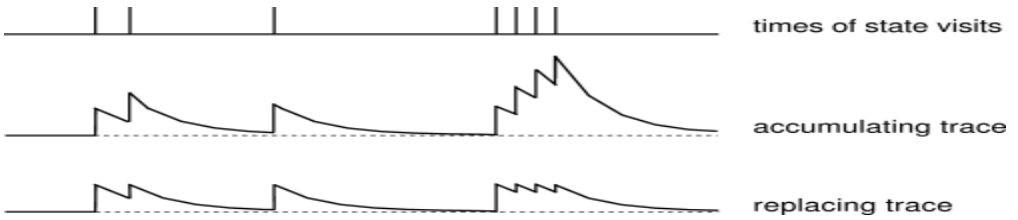
gdzie $e_t(s, a)$ – odrębne ślady aktywności, uaktualniane jak ślady dla krytyka.

# Ślady wymieniane

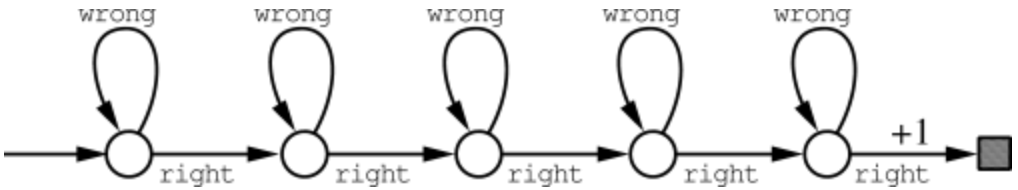
Do metody wartościowania strategii często wprowadza się modyfikację, polegającą na uaktualnianiu śladów aktywności według reguły:

$$e_t(s) = \begin{cases} \gamma\lambda e_{t-1}(s) & \text{if } s \neq s_t; \\ 1 & \text{if } s = s_t. \end{cases}$$

co można zilustrować tak:




Motywacją dla takiego uaktualniania może być zadanie:




Pamiętamy, że algorytm TD(1) *off – line* ze zwykłymi śladami aktywności – to dokładnie *every – visit MC*. Okazuje się, że dla śladów wymienianych – jest to *first – visit MC*.

Stosuje się też odpowiednią modyfikację dla algorytmów uczenia się strategii (Sarsa, Q) polegającą na takim uaktualnianiu śladów:

$$e_t(s,a) = \begin{cases} 1 + \gamma\lambda e_{t-1}(s,a) & \text{if } s = s_t \text{ and } a = a_t; \\ 0 & \text{if } s = s_t \text{ and } a \neq a_t; \\ \gamma\lambda e_{t-1}(s,a) & \text{if } s \neq s_t. \end{cases} \quad \text{for all } s, a$$

# A co z szybkością działania ?

Metody TD(λ) potrafią w mniejszej ilości kroków niż poprzednie metody dojść do dobrych wyników, tylko że tutaj każdy krok, podobnie jak w DP, jest poruszaniem się po całym S, albo (S, A).

Są tu jednak drogi ratunku:

• uaktualnianie tylko kilku ostatnich stanów, czy też par stan – akcja, dla których ślad aktywności powyżej pewnego progu

• stosowanie funkcji wartości stanu nie w postaci tablicowej, lecz jako sparametryzowanej modyfikowalnymi wagami funkcji, która na wejściu dostaje pewne cechy danego stanu (kombinacji tych cech dużo mniej niż stanów), na wyjściu zwraca przybliżoną wartość danego stanu ( dostosowuje się do uczenia takiej funkcji w specjalny sposób dotychczasowe metody).