Spodbujevano učenje

prof. dr. Marko Robnik-Šikonja
oktober 2015

Ideja

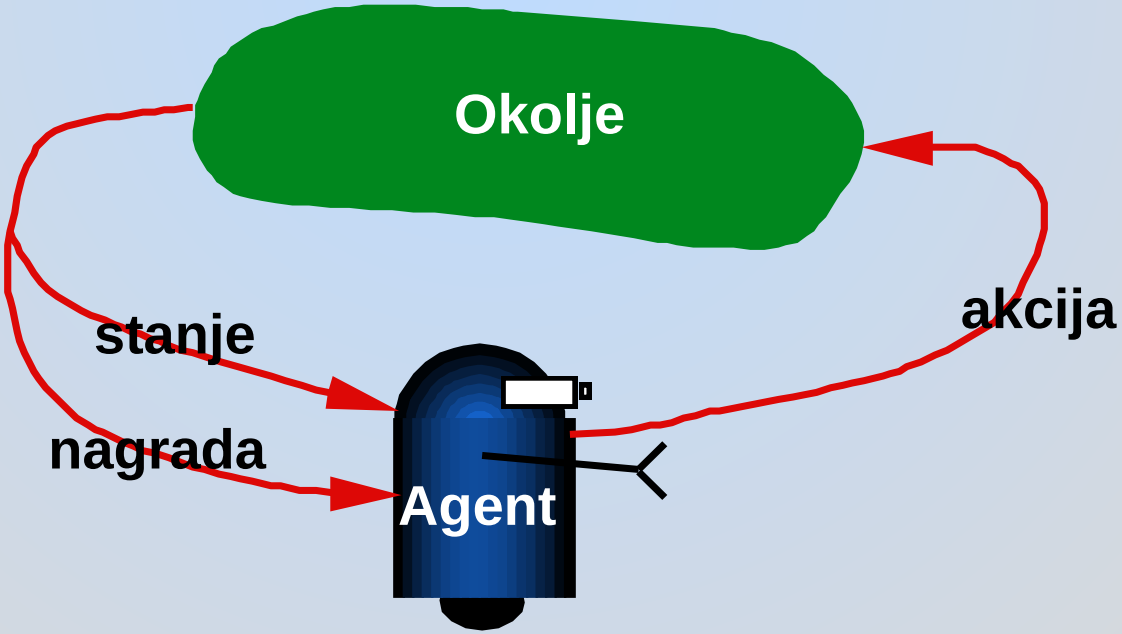
- reinforcement learning (RL), dvomljivo poimenovanje
- izhaja iz psihologije, teorija behaviorizma
- agent se v okolju uči na podlagi svojih akcij
- od okolja dobiva odziv (nagrado, kazen), ki pa ni vedno takojšen
- poskuša izdelati strategijo (policy), ki mu bi omogočila doseči cilje
- primer: igraš igro, za katero ne poznaš pravil, čez 100 potez ti soigralec sporoči: izgubil si

Reference

- R. S. Sutton and A. G. Barto: Reinforcement Learning: An Introduction (tudi nekaj prosojnic)
- Kaelbling, Leslie P.; Michael L. Littman; Andrew W. Moore (1996). "Reinforcement Learning: A Survey". Journal of Artificial Intelligence Research 4: 237–285.
- Busoniu, Lucian; Robert Babuska ; Bart De Schutter ; Damien Ernst (2010). Reinforcement Learning and Dynamic Programming using Function Approximators. Taylor & Francis CRC Press.

Agent

- deluje v času
- neprestano se uči in načrtuje
- vpliva na okolje
- okolje je negotovo in stohastično



Kaj je RL?

- algoritem poskuša na stanja v okolju reagirati tako, da bo maksimiziral nagrado
- agent sam odkrije, katere akcije v določenem stanju prinesejo največjo nagrado
- poskuša doseči cilj: izbira med majhnimi kratkoročnimi in morebitnimi večjimi dolgoročnimi nagradami
- agent si lahko o okolju izdela svojo predstavo (model)
- Izvršene akcije vplivajo na naslednje akcije in tudi na naslednje nagrade
- ne gre za nadzorovano učenje pač pa za poskušanje, raziskovanje, povratno informacijo

Uspehi RL

- robotski nogomet
- upravljanje s premoženjem
- dinamično prirejanje kanalov v mobilnih komunikacija
- nadzor nad dvigali, industrijskimi kontrolerji in roboti
- roboti: navigacija, premikanje, prijemanje
- igre: backgammon (TD-Gammon,Jellyfish)
- ...

Dilema: raziskovati ali izkoriščati

- exploration and exploitation
- ali naj agent izkorišča to kar že ve, da bi maksimiziral nagrado, ali naj preizkuša nova stanja

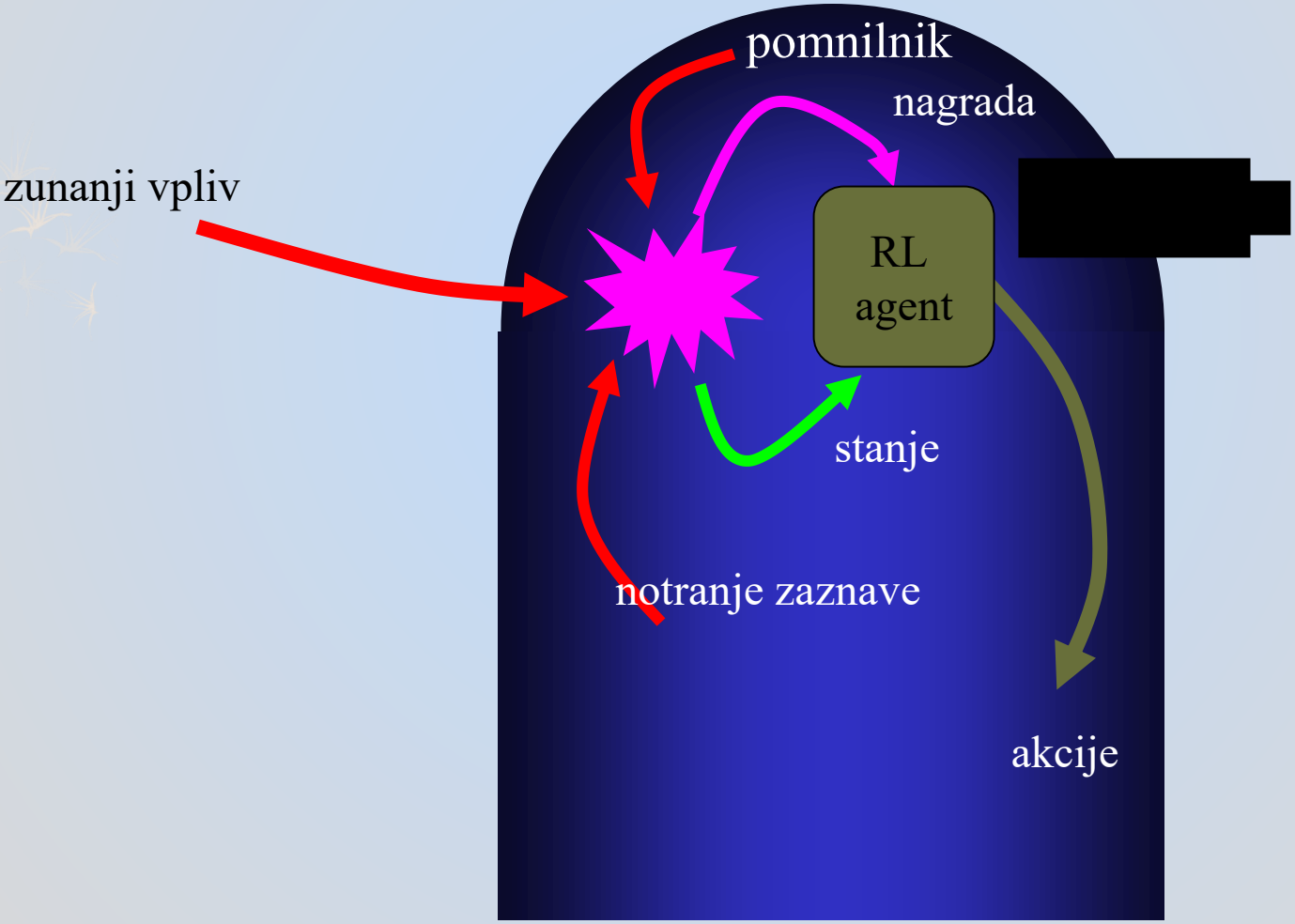
Komponente RL 1/2

- **Strategija**: kaj narediti
  - definira agentove izbire in akcije v danem trenutku
  - predstavljeno z npr. pravili ali tabelo
  - lahko rezultat iskanja, načrtovanja, stohastična,…
- **Nagrada:** povratna informacija iz okolja, agent jo poskuša maksimizirati

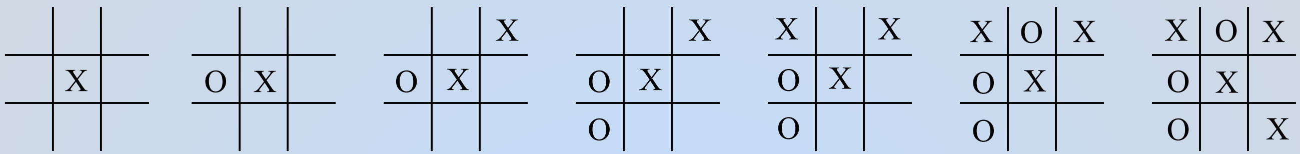
Komponente RL 2/2

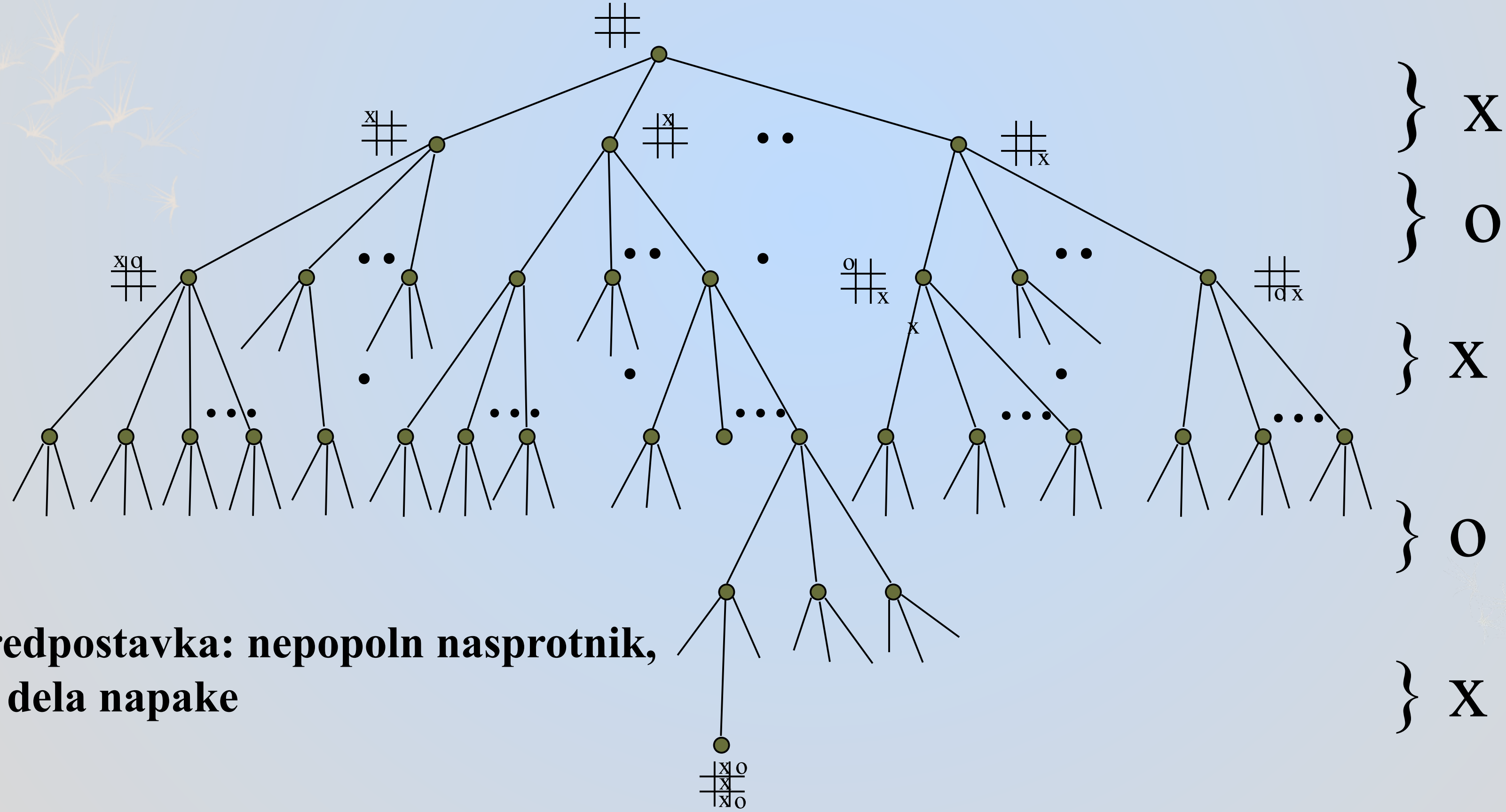
- **Vrednost stanja**: notranje vrednotenje stanja
  - opisuje agentovo pričakovanje, kaj lahko dolgoročno pričakuje v danem stanju
  - implicitno vključuje tudi ocene naslednjih stanj
  - vrednosti stanj se poskušamo naučiti; z večkratnim ponavljanjem (vzorčenjem) jih agent poskuša zanesljivo oceniti
- **Model**: notranji model okolja
  - neobvezna komponenta
  - model omogoča agentu, da ocenjuje vrednosti stanj in akcije, ne da bi jih dejansko izvedel

Shema agenta



Primer: križci in krožci





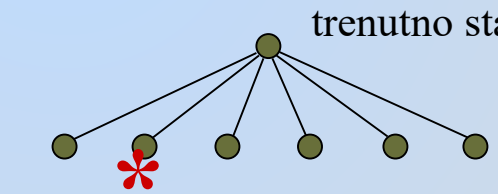
predpostavka: nepopoln nasprotnik,
ki dela napake

RL pristop

1. Tabela z ocenami stanj:

Stanje	$V(s)$ – ocenjena verjetnost zmage	
	0.5	?
	0.5	?
⋮	⋮	
	1	zmaga
⋮	⋮	
	0	poraz
⋮	⋮	
	0	neodločeno

2. Igraj mnogo iger
za izbiro potez poglej
en korak naprej



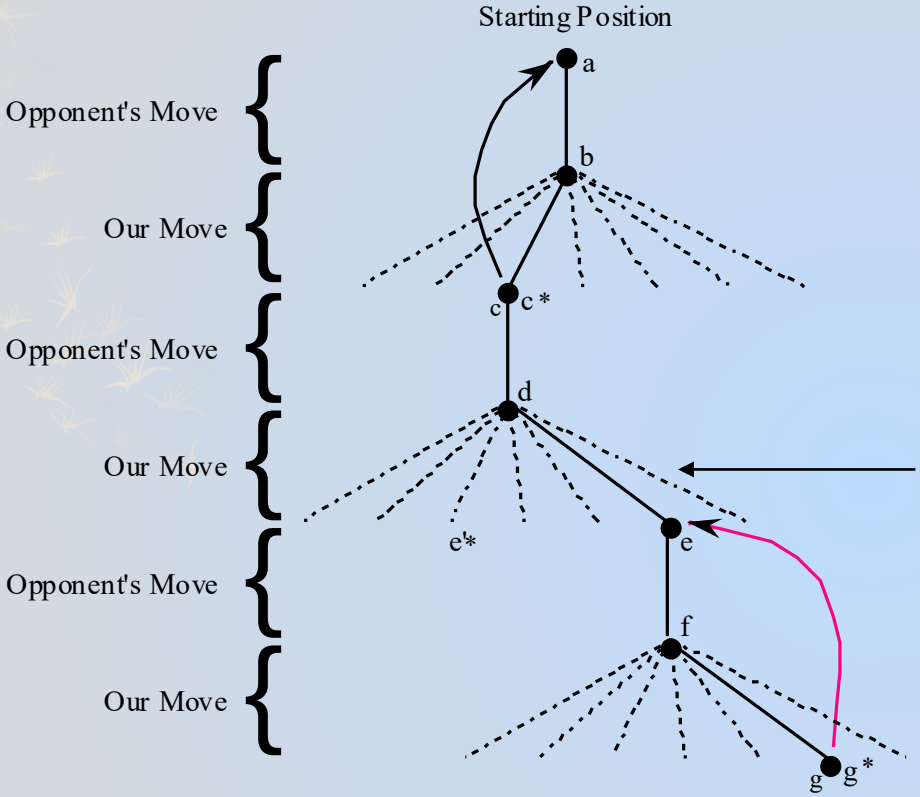
požrešno izberi naslednje stanje z največjo ocenjeno verjetnostjo zmage (maksimalni $V(s)$)

včasih, npr. 10% izberi potezo naključno (raziskovanje)

Križci in krožci

- majhno, končno število stanj, lahko pregledamo vse
  - RL ni omejen na končno število stanj; pri neskončnem ali zelo velikem številu stanj generiramo le tista na katera naletimo in uporabimo pri rešitvi
- gledamo lahko korak vnaprej in požrešno izbiramo
- RL ni omejen na igre ali na odgovor nasprotnika

RL učenje



“raziskovalna” poteza

s – stanje pred požrešno potezo

s' – stanje po požrešni potezi

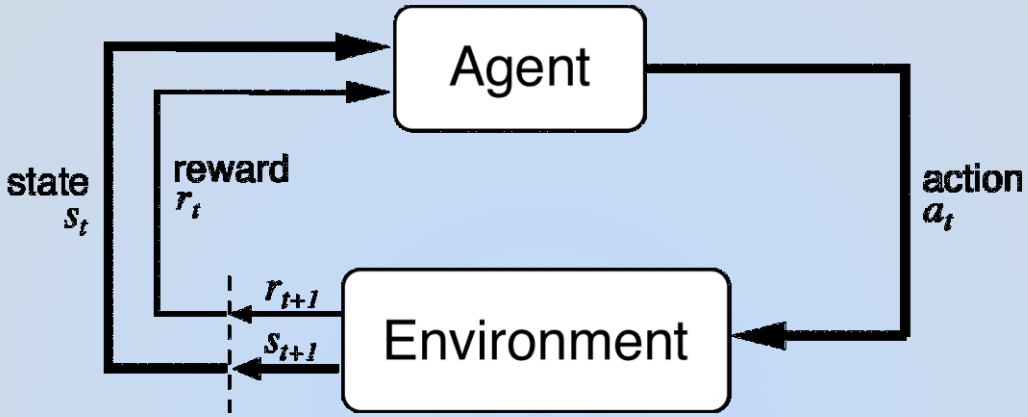
Popravimo vsako $V(s)$ glede na $V(s')$:

$$V(s) \leftarrow V(s) + \alpha\left[V(s') - V(s)\right]$$

kjer je parameter α (velikost koraka)

majhna vrednost npr. 0.1

Formalizacija RL



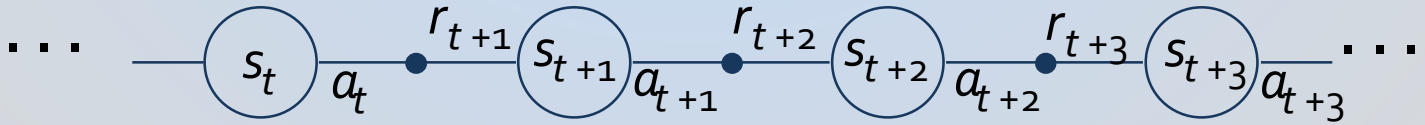
iterakcija z okoljem poteka v diskretnih korakih : $t = 0,1,2,\ldots$

agentovo stanje v koraku t $\quad s_t \in S$

agent v koraku t izvede akcijo : $a_t \in A(s_t)$

dobi nagrado : $r_{t+1} \in \Re$

in preide v naslednje stanje : s_{t+1}



Agentova strategija

Strategija v koraku t, π_t :

preslikava iz stanja v verjetnost akcije

$\pi_t(s,a) = \text{verjetnost, da je } a_t = a, \text{ če je } s_t = s$

$\pi^* = \text{optimalna strategija}$

V preslika stanje v njegovo vrednost

$V^\pi(s) = \text{vrednost stanja } s \text{ pri strategiji } \pi$

- kako agent spreminja strategijo glede na svoje izkušnje
- cilj: dolgoročno pridobiti kar največjo nagrado

Okolje

- nedeterministično
- stacionarno: predpostavimo, da prehode med stanji in nagrade vedno generira enaka verjetnost (to ne pomeni statično!)
- nagrada prihaja iz okolja, ker je agent ne more poljubno spreminjati
- agent ima predstavo o okolju, a ga ne more poljubno spreminjati

Stanja in akcije

- akcije so lahko poljubne interakcije z okoljem, npr. nizkonivojske (spreminjanje napetosti motorja), ali višjenivojske (kupi delnico, premakni figuro)
- stanja so lahko konkretne meritve na podlagi okolja ali pa abstraktne, simbolične, subjektivne (npr.izgubljen)

Nagrada

- cilj določa, kaj želimo doseči in ne kako
- agent lahko eksplicitno in večkrat izmeri svoj uspeh
- maksimiziramo kumulativno vsoto delnih nagrad
- dovolj fleksibilno

Optimalno obnašanje agenta

- kakšno je optimalno obnašanje agenta
- nagrade od časa t naprej: r_{t+1}, r_{t+2}, r_{t+3}, ...
- želimo maksimizirati pričakovano nagrado $E(R_t)$
- tri najbolj razširjene definicije
  - končen horizont, smiselno pri epizodnih problemih (problem lahko razbijemo na epizode, npr. prehod skozi labirint, robot prenese predmet, ...)
  - neskončen horizont (delovanje agenta se ne konča, a bližnja stanja so pomembnejša kot bolj oddaljena)
  - pričakovana povprečna nagrada

Končen horizont

- v času *t* agenta zanima največ *h* stanj vnaprej
- nagrade v tem času so r_{t+1}, r_{t+2}, r_{t+3}, …, r_{t+h}
- $R_t = r_{t+1} + r_{t+2} + r_{t+3} + \ldots + r_{t+h}$
- maksimiziramo pričakovano nagrado v tem obdobju

$$\max E(R_t) = \max E(\sum_{k=1}^{h} r_{t+k})$$

Stategije za končen horizont

- dve strategiji:
  - h-koračna optimalna strategija: na 1. koraku naredi akcijo, ki je najboljša ob predpostavki, da lahko naredi še h-1 akcij, ...
  - h-premično koračna strategija: na vsakem koraku naredi akcijo, ki je najboljša ob predpostavki, da lahko naredi še h akcij
- omejenost pogleda vnaprej
- primernost končnega horizonta: epizodne naloge, npr. pot skozi labirint

Neskončen horizont

- delovanje agenta nima naravnega konca, a bližnja stanja so pomembnejša od bolj odaljenih
- agenta optimizira dolgoročno zaporedje nagrad
- nagrade v prihodnosti se geometrijsko zmanjšujejo
- nagrade: $R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4}$, ... za $0 < \gamma < 1$
- γ (discount factor) lahko interpretiramo kot obresti, način kako omejimo neskončno vsoto, verjetnost preživetja še enega koraka, kratkovidnost/daljnovidnost

$$\max E\left(\sum_{k=0}^{\infty} \gamma^k r_{t+k+1}\right), \quad 0 < \gamma < 1$$

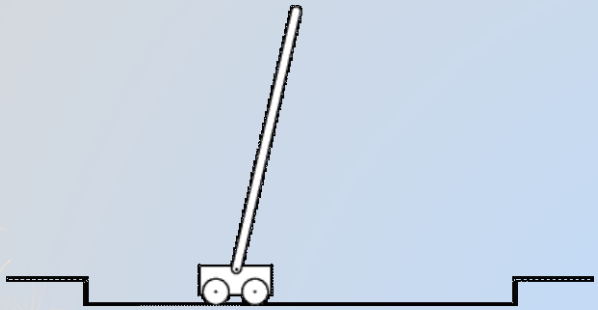
Pričakovana povprečna nagrada

- agenta optimizira dolgoročno povprečno nagrado

$$\lim_{h\to\infty} E\left(\frac{1}{h}\sum_{k=1}^{h} r_{t+k}\right)$$

- slabost: ne loči med bližnjimi in bolj oddaljenimi nagradami

Primer: voziček s palico

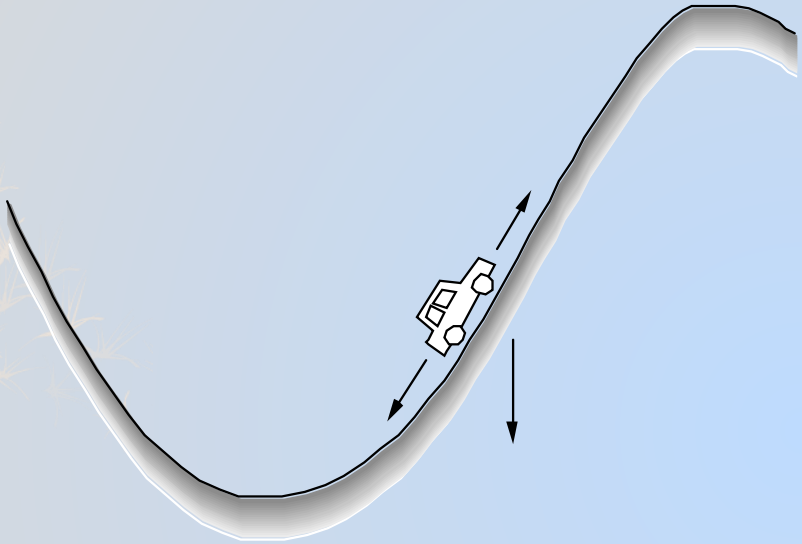


palica med premikanjem ne sme pasti

Epizodna naloga: dokler palica ne pade
nagrada = +1 za vsak korak pred padcem
uspeh = število korakov pred padcem

Ponavljujoča se naloga s padajočo nagrado
nagrada = -1 ob padcu, 0 sicer
uspeh = γ^k za k korakov pred padcem ($0 < \gamma < 1$)

Primer: klanec



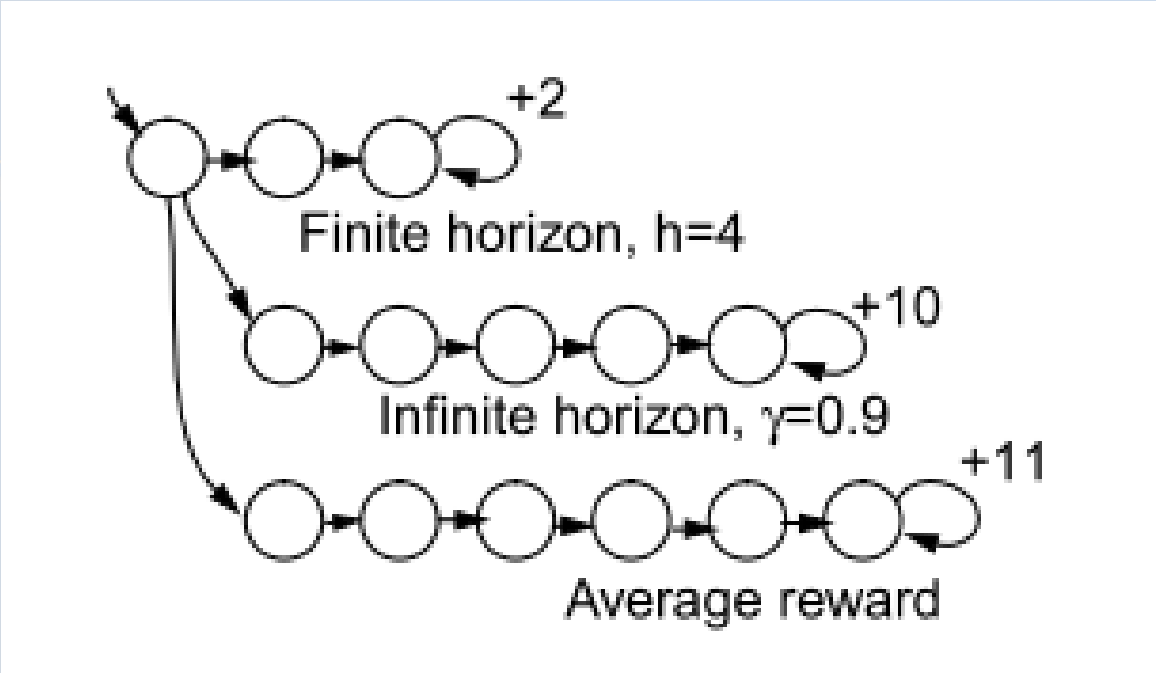
Premakni se na vrh hriba, kakor hitro je mogoče

nagrada $= -1$ za vsak korak, ko nismo na vrhu hriba

$\Rightarrow$ uspeh $=-$ št. korakov preden dosežemo vrh

Uspeh maksimiciramo, če minimiziramo število korakov, da pridemo na vrh hriba.

Primerjava obravnavanja nagrad



Kriteriji za uspešnost učenja

- konvergenca: nekateri algoritmi z dokazano konvergenco k optimalni rešitvi
- hitrost konvergence:
  - hitrost konvergence k skoraj optimalni rešitvi
  - stopnja optimalnosti po določenem času
  - parameter (stopnja optimalnost ali čas)
- razlika do optimalne strategije (regret): vsota razlik med optimalno strategijo in strategijo, ki jo je našel učni algoritem

Markovski RL problem

- stanje je markovsko, če je naslednje stanje odvisno le od trenutnega
- npr. šah: trenutna pozicija vsebuje vso potrebno informacijo za nadaljevanje, ni nam potrebno poznati zgodovine potez
- koristna aproksimacija, ki poenostavi učne algoritme; celo če okolje modeliramo

$$\Pr\{s_{t+1}=s', r_{t+1}=r \mid s_t, a_t, r_t, s_{t-1}, a_{t-1}, \ldots, r_1, s_0, a_0\} =$$

$$\Pr\{s_{t+1}=s', r_{t+1}=r \mid s_t, a_t\}$$

za vse s', r in pretekle $s_t, a_t, r_t, s_{t-1}, a_{t-1}, \ldots, r_1, s_0, a_0$.

Markovski odločitveni proces

- če velja Markovska lastnost, gre za MDP (Markov decision process)
- če je množica stanj in akcij končna, imamo končni MDP, kar predstavlja napomembnejši primer RL
- definiramo
  - množico stanj in akcij
  - verjetnosti prehodov za en korak

$$T(s,a,s') = P(s_{t+1} = s' \,|\, s_t = s, a_t = a) \text{ za vse } s, s' \in S, a \in A(s).$$

  - verjetnosti nagrad:

$$\mathbf{R}(s,a) = E(r_{t+1} \,|\, s_t = s, a_t = a) \text{ za vse } s \in S, a \in A(s).$$

Primer končnega MDP

Čistilni robot v parku zbira odpadle pločevinke

- na vsakem koraku se robot odloča ali naj 1) aktivno išče pločevinke 2) počaka, da mu kdo prinese pločevinko 3) se vrne v bazo in polni
- iskanje prinese večjo nagrado, a hitreje prazni baterijo; če mu zmanjka baterije med iskanjem, ga je treba rešiti, kar prinese veliko kazen
- odločitev glede na trenutno stanje energije: `high`, `low`.
- nagrada = število zbranih pločevink

Čistilni robot MDP

$S = \{\texttt{high}, \texttt{low}\}$

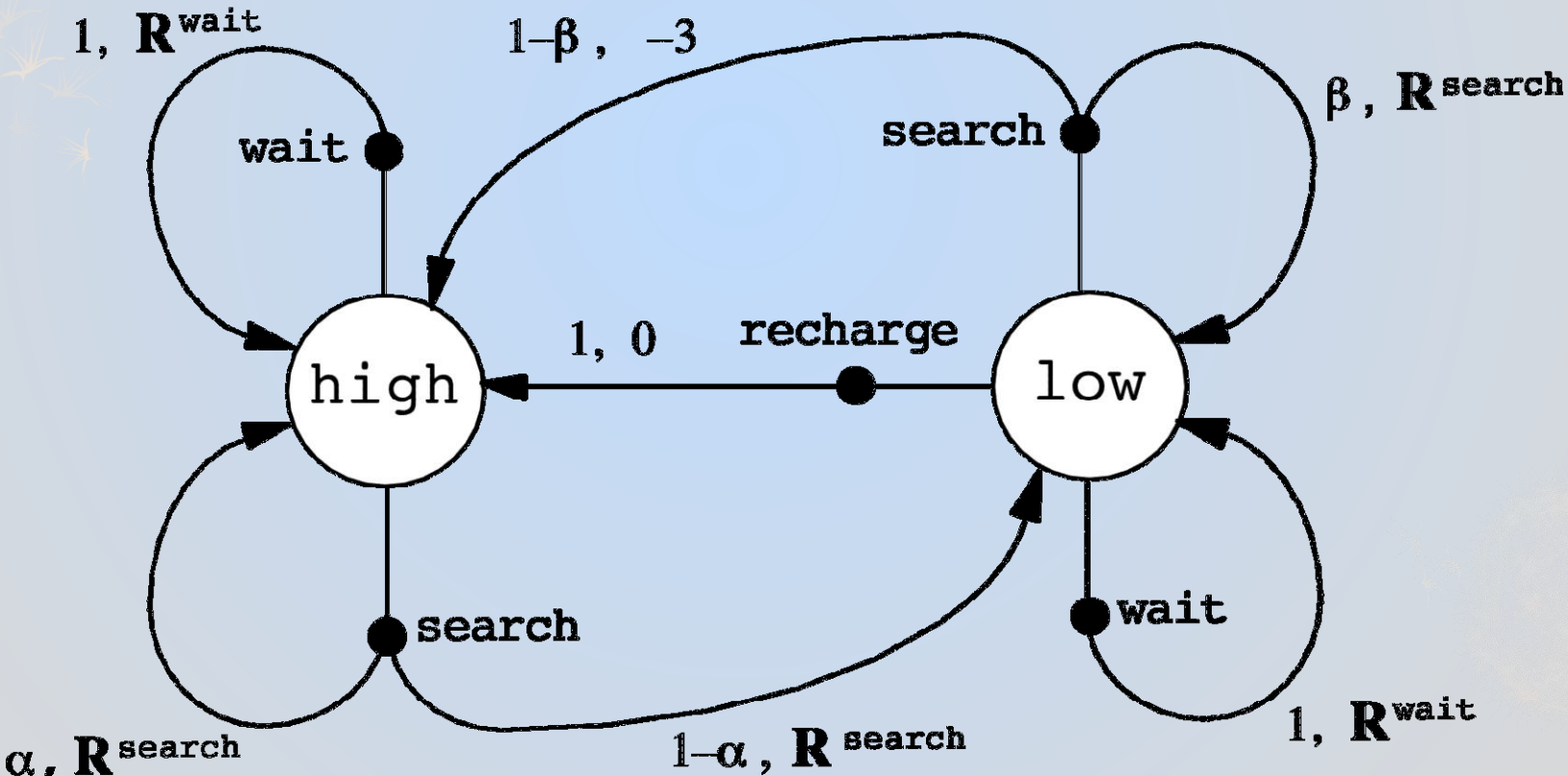
$A(\texttt{high}) = \{\texttt{search}, \texttt{wait}\}$

$A(\texttt{low}) = \{\texttt{search}, \texttt{wait}, \texttt{recharge}\}$

$\mathbf{R}^{\texttt{search}}$ = expected no. of cans while searching

$\mathbf{R}^{\texttt{wait}}$ = expected no. of cans while waiting

$\mathbf{R}^{\texttt{search}} > \mathbf{R}^{\texttt{wait}}$



Funkcija vrednosti za MDP

- Vrednost stanja je pričakovana nagrada, če začnemo iz danega stanja in sledimo agentovi strategiji π:

$$V^{\pi}(s) = E_{\pi}\left\{R_t \middle| s_t = s\right\} = E_{\pi}\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \middle| s_t = s\right\}$$

- Definiramo funkcijo Q – vrednost akcije v danem stanju in pri dani strategiji: pričakovana nagrada

$$Q^{\pi}(s,a) = E_{\pi}\left\{R_t \middle| s_t = s, a_t = a\right\} = E_{\pi}\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \middle| s_t = s, a_t = a\right\}$$

Izračun V in Q

- vrednosti $V^{\pi}(s)$ in $Q^{\pi}(s,a)$ ocenimo na podlagi izkušenj
- teoretično v limiti konvergirajo v pravo vrednost
- agent si za vsako stanje/akcijo zapomni povprečje
- pri velikih prostorih stanja generaliziramo
- ocenimo/izračunamo jih z Monte-Carlo učnimi metodami
- lahko jih zapišemo rekurzivno

Bellmanova enačba za strategijo π

Ideja

$$
\begin{aligned}
R_t &= r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} \cdots \\
&= r_{t+1} + \gamma \left(r_{t+2} + \gamma r_{t+3} + \gamma^2 r_{t+4} \cdots \right) \\
&= r_{t+1} + \gamma R_{t+1}
\end{aligned}
$$

Za vrednost

$$
\begin{aligned}
V^{\pi}(s) &= E_{\pi}\left\{ R_t \,\middle|\, s_t = s \right\} \\
&= E_{\pi}\left\{ r_{t+1} + \gamma V\left(s_{t+1}\right) \,\middle|\, s_t = s \right\}
\end{aligned}
$$

Oziroma drugače:

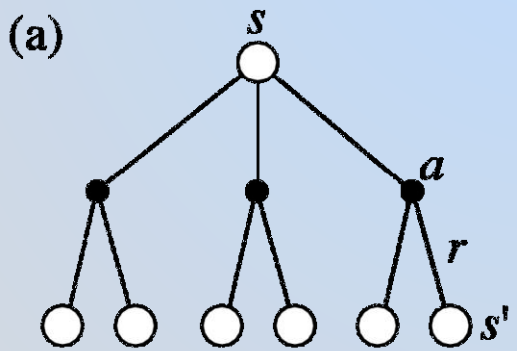
$$
V^{\pi}(s) = \sum_a \pi(s,a) \sum_{s'} T(s,a,s') \left[\mathbf{R}(s,a) + \gamma V^{\pi}(s') \right]
$$

Bellmanova enačba

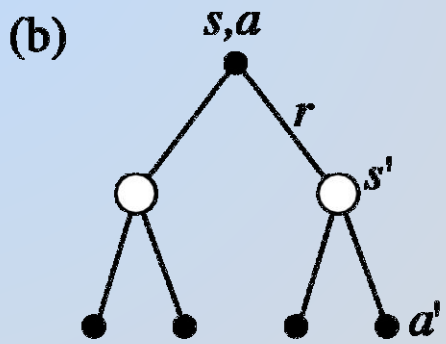
$$V^{\pi}(s) = \sum_{a} \pi(s,a) \sum_{s'} T(s,a,s') \left[\mathbf{R}(s,a) + \gamma V^{\pi}(s') \right]$$

- Množica linearnih enačb, po ena za vsako stanje z eno samo rešitvijo.
- Dinamično programiranje

Vračanje vrednosti:



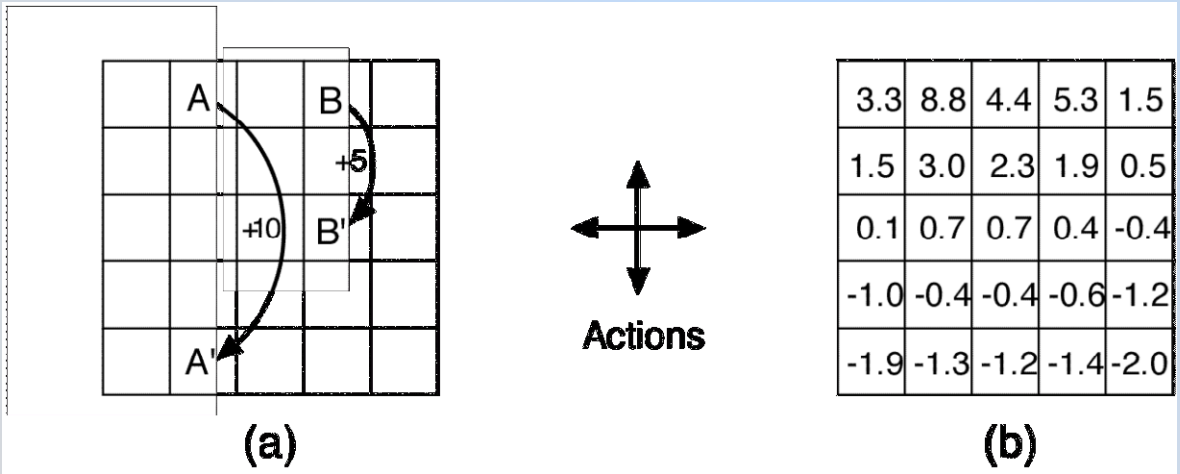
za V^{π}



za Q^{π}

Primer: kvadratna mreža

- Akcije: S,J, V, Z
- če bi agent padel iz mreže ostane na istem mestu, z nagrado –1
- druge akcije imajo nagrado 0, razen v točkah A in B



strategija: v vse smeri z enako verjetnostjo

rešitev Bellmanovih enačb za $\gamma = 0.9$

Izbor strategije pri danem modelu

- denimo, da model poznamo
- model okolja pri MDP je podan s T(s,a,s') in R(s,a)
- optimalna vrednost stanja $V^*(s)$

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

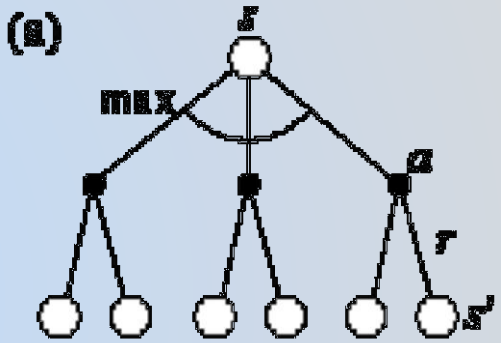
- v nadaljevanju predpostavimo neskončen horizont zaradi lažje izpeljave

Bellmanove enačbe za optimalno strategijo

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

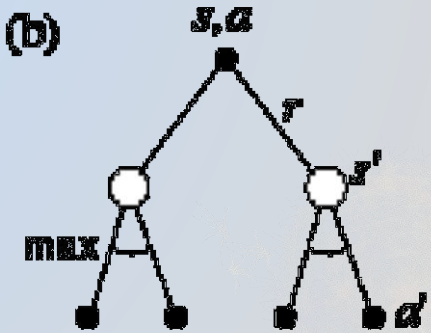
$$= \max_{a \in A(s)} E\left\{ r_{t+1} + \gamma V^*(s_{t+1}) \middle| s_t = s, a_t = a \right\}$$

$$= \max_{a \in A(s)} \sum_{s'} T(s,a,s') \left[\mathbf{R}(s,a) + \gamma V^*(s') \right]$$



$$Q^*(s,a) = E\left\{ r_{t+1} + \gamma \max_{a'} Q^*(s_{t+1}, a') \middle| s_t = s, a_t = a \right\}$$

$$= \sum_{s'} T(s,a,s') \left[R(s,a) + \gamma \max_{a'} Q^*(s',a') \right]$$



Reševanje Bellmanovih optimalnih enačb

- Če želimo najti optimalno strategijo z rešitvijo Bellmanovih enačb moramo za naš problem izpolnjevati Markovski pogoj, poznati moramo natančno dinamiko okolja ter imeti na voljo zadosti pomnilnika in časa
- zahtevnost reševanja je z dinamičnim programiranjem polinomska glede na število stanj (npr. backgammon okoli 10^{20}), osredotočimo se torej na bolj verjetna stanja, na katera naletimo med iskanjem
- za večino zanimivih problemov poskušamo najti približne rešitve
- številne RL metode so v bistvu približne metode za reševanje Bellmanovih optimalnih enačb

Iteracija vrednosti

void valueIteration() {
  poljubno inicializiraj vrednosti $V(s)$
  do{
    foreach ($s \in S$) {
      foreach ($a \in A$) {
        $Q(s,a) = R(s,a)+\gamma \Sigma_{s' \in S} T(s,a,s')V(s')$
        $V(s) = \max_a Q(s,a)$
      }
    }
  } **while** (! zadovoljen) ;
}

algoritem nazaj (od končnih stanj) popravlja vrednosti

Iteracija vrednosti: konvergenca

- Izrek: če je razlika dveh zapored izračunanih vrednosti stanja manjša od ε , je vrednost požrešne strategije, ki uporablja te ocene, v vsakem stanju od optimalne manjša za največ $2\varepsilon\gamma/(1-\gamma)$.
- lahko uporabimo kot ustavitveni pogoj
- robusten pristop, saj lahko vrednosti stanjem prirejamo tudi asinhrono in vzporedno, konvergenca pa se ne spremeni

Iteracija strategije

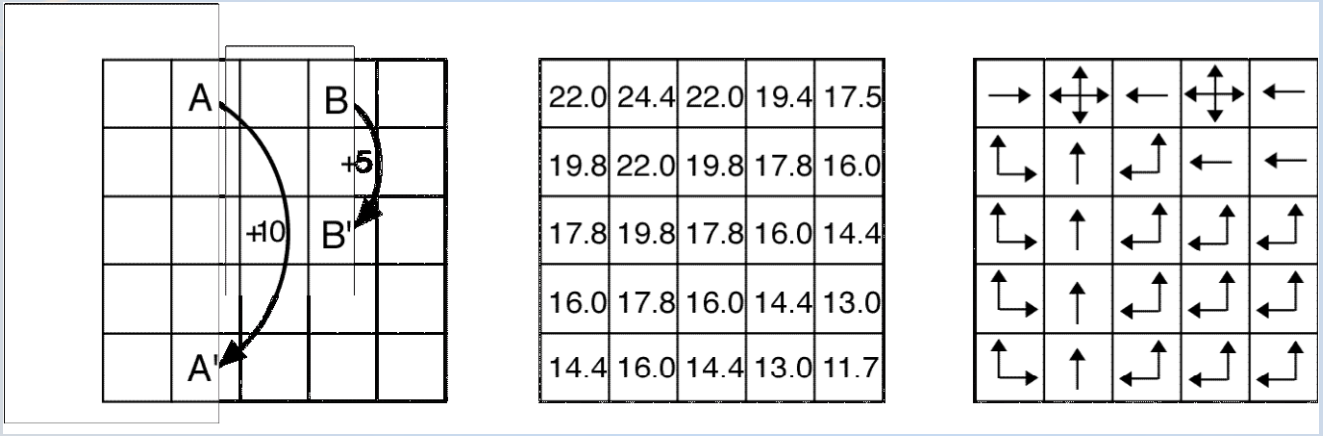
void policyIteration() {
  izberi poljubno strategijo π'
  do {
    $\pi = \pi'$
    izračunaj vrednost strategije π tako, da rešiš linearne enačbe
      $V_\pi(s) = R(s, \pi(s)) + \gamma \sum_{s' \in S} T(s, \pi(s), s') V_\pi(s')$;
    izboljšaj strategijo v vsakem stanju
      $\pi'(s) = \arg\max_a R(s, a) + \gamma \sum_{s' \in S} T(s, a, s') V_\pi(s')$;
  } **while** ($\pi \neq \pi'$) ;
}

- Ko vrednost dobimo z rešitvijo linearnih enačb, preverimo ali lahko strategijo izboljšamo s spremembo prve akcije. Če ne moremo spremeniti strategije v nobenem stanju, je dobljena strategija zagotovljeno optimalna.

Uporaba optimalnih V* in Q*

S poznavanjem V* je optimalna kar požrešna strategija.
Če imamo V*, nam pogled naprej za en korak da optimalne rezultate.

Primer:



V^* $? \pi^*$

Približne metode reševanja

- učenje s časovnimi razlikami (ni potreben model, inkrementalne, težavne za analizo)
- dinamično programiranje (matematično dobro podprto, zahteva popoln in natančen opis sveta)
- Monte Carlo metode (ne zahtevajo modela, konceptualno preproste, neprimerne za inkrementalen izračun, vzorčijo kompletne trajektorije v interakciji z okoljem ali modelom okolja)
- razlike glede učinkovitosti in hitrosti konvergence

Učenje s časovnimi razlikami

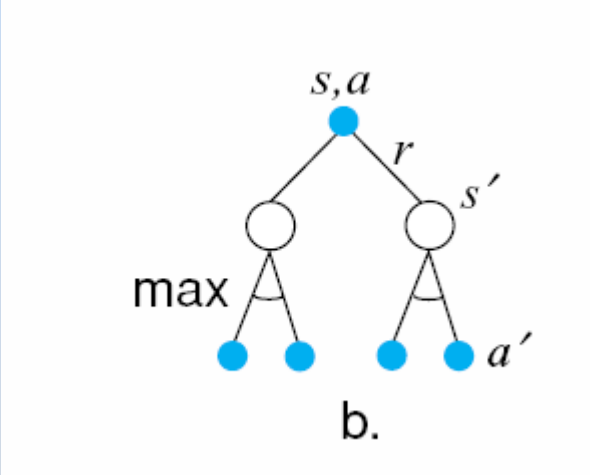
- nazaj na prejšnja stanja prenesemo nekaj razlike z naslednjimi stanji
  $V(s_t) = V(s_t) + c(V(s_{t+1}) - V(s_t))$
- c je parameter, ki se med učenjem postopno zmanjšuje in omogoča konvergenco

Q učenje

- Watkins, 1989
- pogosto uporabljena varianta časovnih razlik
- za en korak naprej

$$Q(s_t,a_t) = (1\text{-}c)\, Q(s_t,a_t) + c(r_{t+1} + \gamma \max_a Q(s_{t+1},a) - Q(s_t,a_t))$$

$0 <= c, \gamma <= 1$



Q učenje

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., ε-greedy)
        Take action a, observe r, s'
        $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$
        $s \leftarrow s'$;
    until s is terminal

Pregled metod