Osnove podržanog učenja

Ozretić, Petar

PRIRODOSLOVNO–MATEMATIČKI FAKULTET
SVEUČILIŠTA U SPLITU

PETAR OZRETIĆ

OSNOVE PODRŽANOG UČENJA

DIPLOMSKI RAD

Split, rujan 2022.

PRIRODOSLOVNO–MATEMATIČKI FAKULTET
SVEUČILIŠTA U SPLITU

ODJEL ZA MATEMATIKU

OSNOVE PODRŽANOG UČENJA

DIPLOMSKI RAD

Student:
Petar Ozretić

Mentor:
izv. prof. dr. sc.
Saša Mladenović

Split, rujan 2022.

Uvod

Učenje uz pomoć okoline prati ljudska bića od prvih dana, kad se beba igra, maše rukama, promatra svijet itd., nema izričitog učitelja, ali ima direktnu senzomotornu vezu sa svojom okolinom. Ova veza je bogat izvor informacija o uzrocima i posljedicama, akcijama i reakcijama, i o tome što treba napraviti da bi došli do svog cilja, e.g. *ako zaplačem, promijenit će mi pelenu.*

Interakcija s okolinom je forma učenja koja prati čovjeka tijekom cijeloga života - kada npr. čovjek uči voziti automobil, mora biti izrazito pozoran na svoju okolinu i kako ona reagira na ono što ona čini - *ako vozači na cesti blicaju i trube, vjerojatno sam ja taj koji vozi krivom stranom autoceste.*

Podržano učenje (PU) je učenje postupaka koji će maksimizirati nekakvu numeričku nagradu. Učeniku nitko ne govori što bi trebalo učiniti - sam mora isprobati radnje i otkriti koja od njih mu donosi najveću nagradu. Najzanimljiviji i najteži su slučajevi kada neka radnja ne utječe samo na trenutnu nagradu, nego i na iduću situaciju, a time i na sve buduće nagrade. Ova dva svojstva - *učenje iz pokušaja i pogrešaka* te *odgođena nagrada*, odnosno dalekosežne posljedice radnji - su razlikovne karakteristike podržanog učenja u odnosu na druge paradigme korištene u umjetnoj inteligenciji.

Dok je razlika između podržanog i nenadziranog učenja - koje pokušava pronaći skrivenu strukturu u skupu podataka - očita, granicu između podržanog i nadziranog učenja je puno teže razlučiti. Nadzirano učenje je učenje iz skupa za treniranje koji sadrži označene primjere koje dobijamo od sveznajućeg "nadziratelja". Svaki primjer je dan kao opis situacije zajedno s ispravnom radnjom koju bi trebalo poduzeti u toj situaciji, najčešće je to određivanje kojoj kategoriji ta situacija pripada. Cilj ove vrste učenja je generalizacija danih odgovora kako bi to kasnije mogli primijeniti na situacije koje se ne nalaze u skupu za treniranje. Iako je nadzirano učenje jako važno, nije dostatno da obuhvati učenje iz interakcija s okolinom jer je jako često nepraktično izraditi toliko primjera koji bi pokrili apsolutno sve situacije. Na ovom neistraženom teritoriju treba biti sposoban učiti iz vlastitog iskustva.

Još jedna razlikovna karakteristika koju se ne susreće kod drugih vrsta učenja je kompromis između istraživanja i iskorištavanja trenutnog znanja. Da bi dobio što veću nagradu učenik mora koristiti one radnje koje je već nekad u prošlosti isprobao i za njih dobio dobru nagradu. Ali da bi uopće otkrio koje su to radnje mora isprobati radnje koje do sada nije. Učenik mora iskorištavati svoje trenutno znanje da bi dobio nagradu, ali mora i nastaviti istraživati da bi u budućnosti mogao još bolje birati. Vrijedi uočiti da iako je dilema između istraživanja i iskorištavanja desetljećima bila tema brojnih matematičkih istraživanja, ipak i dalje nije razriješena.

Još jedno bitno svojstvo podržanog učenja koje je za rad važno spomenuti - za razliku od drugih vrsta učenja, i općenito prakse u programiranju, gdje se veći problem razbija na više manjih - podržano učenje promatra problem isključivo u cjelini. *Nebitno je ako na vozačkom ispitu cijelo vrijeme vozi savršeno, ako se na kraju vozač zabije u zid.*

Sadržaj

Poglavlje 1

Osnovni pojmovi podržanog učenja

1.1 Agent i okolina

Da bi se prilagodili standardnoj nomenklaturi PU-a, učenika, tj. program koji donosi odluke nazivamo *agent*, a *okolina* je sve ono izvan našeg agenta s čime agent vrši interakciju. Radnju ili odluku koju agent vrši ili donosi nazivamo *akcija*[1]. Osim ovih, još su četiri podelementa PU-a: strategija, nagrada/nagradni signal, vrijednosna funkcija (en. *value function*), te model okoline.

Strategija definira kako se agent ponaša u određenoj situaciji. Ugrubo govoreći može se reći da je *strategija* preslikavanje iz stanja okoliša u skup svih akcija koje se mogu poduzeti u tim stanjima. Strategija može biti i stohastička - u smislu da poručuje samo kolika je vjerojatnost da će agent poduzeti neku akciju u određenoj situaciji.

[1]U literaturi se za pojmove *agent*, *okolina* i *akcija* znaju koristiti i više inženjerski pojmovi *kontroler*, *kontrolirani sustav* i *kontrolni signal* redom.

Pomoću *nagradnog signala* definira se cilj problema PU-a. U svakom koraku okolina šalje agentu broj koji se naziva *nagrada*. Agentov je jedini cilj maksimizirati ukupnu nagradu - nagradnim signalom se onda može definirati koja su ponašanja poželjna, a koja ne. Nagradni signal je temeljni element prilagođavanja strategije agenta - ako neka akcija daje malenu nagradu može se prilagoditi strategiju da drugi put u takvoj situaciji agent isproba neku drugu akciju.

Nagradni signal govori što je dobro u datom trenutku, dok *vrijednosna funkcija* kaže što je dobro dugoročno gledano. Drugim riječima, *vrijednost* nekog stanja u kojem se agent nalazi je ukupna količina nagrade koju agent može prikupiti ako se nalazi u tom stanju - znači uzima se u obzir trenutna nagrada koja se dobije u tom stanju i stanja do kojih se može doći iz njega, zajedno sa njihovim nagradama. Upravo se **vrijednost** najviše uzima u obzir kada se vrši procjena odluke. Poželjne su one akcije koje će rezultirati stanjem s najvećom vrijednosti, a ne samo najvećom nagradom jer vrijednost stanja može dugoročno donijeti veću nagradu. Nažalost, vrijednosti je puno teže procijeniti. Dok nagrada proizlazi neposredno iz okoliša, vrijednosti je potrebno procjenjivati iz niza opservacija koje agent učini tokom svog rada. Efikasno procjenjivanje vrijednosne funkcije je najvažniji dio gotovo svih algoritama PU-a.

Model okoline nije nužan element PU-a. Model govori kako će okolina reagirati na akciju agenta - u kojem će se stanju naći i koliku će nagradu dobiti. Modeli zapravo služe za planiranje budućih akcija.

1.2 Markovljev proces odlučivanja

Markovljev proces odlučivanja (MDP) je klasična formalizacija slijednog donošenja odluka kada akcija ne utječe samo na trenutnu nagradu, već i sve sljedeće situacije ili stanja i s njima povezane nagrade.

Definicija 1.1 (MDP) *MDP je uređena trojka* $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{P}_0)$ *gdje je*

$\mathcal{S}$ *(prebrojiv) neprazan skup čije elemente nazivamo stanja;*

$\mathcal{A}$ *(prebrojiv) neprazan skup čije elemente nazivamo akcije;*

$\mathcal{P}_0$ *matrica prijelaza, koja svakom paru* $(s, a) \in \mathcal{S} \times \mathcal{A}$ *pridružuje vjerojatnosnu mjeru povrh* $\mathcal{S} \times \mathbb{R}$ *koju označavamo sa* $\mathcal{P}_0(\cdot|s, a)$.[2]

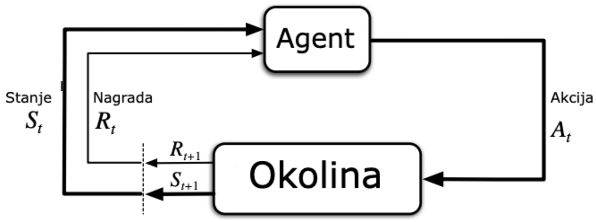
Kao što je već spomenuto MDP je alat za modeliranje problema slijednog donošenja odluka. Ako je dan MDP $\mathcal{M}$, interakcija izgleda ovako: Neka je $t \in \mathbb{N}$ trenutno vrijeme (razina, trenutak), neka su $S_t \in \mathcal{S}$ i $A_t \in \mathcal{A}$ nasumično odabrano stanje sustava i akcija koju će agent poduzeti u vremenu t redom. U idućem koraku, kao posljedicu njegove akcije, agent dobija numeričku nagradu $R_{t+1} \in \mathcal{R} \subset \mathbb{R}$, i nalazi se u novom stanju S_{t+1}. Ovim postupkom MDP i agent nam daju tzv. *putanju*: $S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, ...$

U Markovljevom procesu odlučivanja vjerojatnosti koje daje matrica prijelaza $\mathcal{P}_0$ u potpunosti opisuju dinamiku okoline - tj. vjerojatnosti za sve vrijednosti S_t i R_t ovise o neposredno prethodnom stanju S_{t-1} i akciji A_{t-1} [3] , i samo o njima, uopće se ne obazire na stanja i akcije koje im prethode. Ovo je najbolje promatrati kao svojstvo stanja, a ne cjelokupnog procesa.

[2]što znači da za skup $\mathcal{U} \subset \mathcal{S} \times \mathbb{R}$, $\mathcal{P}_0(\mathcal{U}|s, a)$ govori kolika je vjerojatnost da će iduće stanje i njemu pripadajuća nagrada biti u skupu $\mathcal{U}$, ako se agent trenutno nalazi u stanju s i poduzme akciju a.

[3]Skup $\mathcal{U}$ iz prethodne napomene je u ovom slučaju jednočlan, tj. $\mathcal{U} = \{S_t\} \times \{R_t\}$, pa pišemo $\mathcal{P}_0(S_t, R_t|S_{t-1}, A_{t-1})$.

Stanje u sebi mora sadržavati sve potrebne informacije o prošlim interakcijama agenta i sustava koje imaju utjecaj na budućnost - u tom slučaju kaže se da stanje ima *Markovljevo svojstvo*.



Slika 1.1: Interakcija agenta i okoline u MDP-u [2]

Iz funkcije $\mathcal{P}_0$ može se izračunati sve o okolini što bi moglo biti korisno ili zanimljivo, kao na primjer *matricu prijelaza stanja* $\mathcal{P} : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \to [0, 1]$

$$\mathcal{P}(s'|s, a) = \mathbb{P}[S_t = s'|S_{t-1} = s, A_{t-1} = a] = \sum_{r \in \mathcal{R}} \mathcal{P}_0(s', r|s, a)$$

Također je moguće i *očekivanu nagradu* izraziti kao funkciju stanja i akcije $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$:

$$r(s, a) = \mathbb{E}[R_t|S_{t-1} = s, A_{t-1} = a] = \sum_{r \in \mathcal{R}} r \sum_{s' \in \mathcal{S}} \mathcal{P}_0(s', r|s, a)$$

Kako je već spomenuto, agentov je cilj maksimizirati sveukupnu nagradu - znači ne trenutnu, već kumulativnu, dugoročnu nagradu. Iako se ovo možda čini limitirajuće, u praksi se pokazalo kao fleksibilno i široko primjenjivo. Na primjer ako agent treba naučiti kako pobjeći iz labirinta, a "nagrada" je -1 za svaki korak koji agent napravi prije nego li uspije izaći - na ovaj način agent će nastojati pobjeći iz labirinta što prije.

Budući da agent uvijek nalazi način kako maksimizirati svoju nagradu, a cilj je da agent izvrši određeni zadatak, onda mu za taj zadatak mora biti ponuđena nagrada na način da maksimizirajući nagrade agent usput postigne i željeni cilj. Od iznimne je važnosti da se nagrade uistinu podudaraju s onim što se želi postići - npr. agent koji uči igrati šah bi trebao biti nagrađen samo za pobjeđivanje, ne i za postizanje određenih podciljeva kao što su uzimanje protivničkih figura ili zauzimanje sredine ploče. Nagrađivanjem podciljeva može doći do situacije da agent nađe način da maksimizira svoju nagradu bez postizanja željenog cilja. Npr. može uzeti neprijateljsku figuru pod cijenu gubljenja partije.

Ako je niz nagrada koje se dobiju nakon koraka t označen s $R_{t+1}, R_{t+2}, R_{t+3}...$, koji se točno dio ovoga niza želi maksimizirati? Općenito agent želi maksimizirati *očekivanu dobit* - *eng. expected return*, gdje se dobit , u oznaci G_t, definira kao funkciju niza nagrada. Najjednostavnije je uzeti sumu nagrada $G_t = R_{t+1} + R_{t+2} + R_{t+3} + ... + R_T$, gdje je T zadnji korak. Ovo ima smisla u primjerima koji u sebi po prirodi stvari podrazumijevaju neki zadnji korak, tj. kada se interakciju agenta i okoline može na prirodan način podijeliti u tzv. *epizode*: partija igre, jedan prolaz kroz labirint, bilo koja vrsta ponavljane interakcije. Svaka epizoda završava u posebnom stanju koje nazivamo *terminalno stanje*, nakon čega se vraća na početno stanje (ili na neko od početnih stanja ako ih ima više). Svaka epizoda je neovisna od prethodne, bez obzira kako je ona završila - pobjedom ili porazom.

Iako ovakvi epizodni scenariji pokrivaju puno slučajeva, od posebnog su interesa i kontinuirane interakcije koje se nastavljaju bez jasnog kraja ili zadnjeg koraka. U tom slučaju je $T = \infty$ pa se uvodi koncept *korekcije*, eng. *discounting*. Agent bira akcije na način da suma nagrada s korekcijom bude maksimalna, tj. bira A_t da bi maksimizirao korigiranu dobit

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{1.1}$$

parametar γ, $0 \leq \gamma \leq 1$, nazivamo *korekcijski faktor*.

Korekcijski faktor određuje trenutnu vrijednost budućih nagrada: nagrada koju može dobiti za k koraka je vrijedna za faktor γ^{k-1} manje nego da je primi sada. Ako je $\gamma < 1$ i skup $\mathcal{R}$ omeđen, onda je beskonačna suma u (1.1) isto omeđena. Ako je $\gamma = 0$, onda je taj agent "kratkovidan" i zanima ga samo maksimiziranje trenutne nagrade: zanima ga samo kako odabrati A_t kako bi maksimizirao samo R_{t+1}. Općenito, ovo će spriječiti agenta od dobijanja nekih budućih odgođenih nagrada i smanjiti njegovu ukupnu dobit. Koliko se γ približava 1, to taj agent sve više uzima u obzir buduće nagrade - postaje "dalekovidniji".

1.3 Funkcija vrijednosti

Gotovo svi algoritmi PU-a se temelje na određivanju funkcije vrijednosti - funkcija koja nam govori "koliko je dobro" za agenta da bude u određenom stanju. Vrijednosne funkcije definiramo s obzirom na ponašanje agenta, odnosno *strategije* koju agent slijedi. Formalno, *strategija* je mapiranje koje svakom stanju pridruži distribuciju na skupu akcija[4]. Ako agent slijedi strategiju π u trenutku t, onda nam $\pi(a|s)$ kaže kolika je vjerojatnost da je $A_t = a$ ako je $S_t = s$, tj. ako se u trenutku t agent nalazi u stanju s, kolika je vjerojatnost da će poduzeti akciju a.

Vrijednosna funkcija stanja s u strategiji π, u oznaci $v_\pi(s)$, je očekivana dobit agenta koji počevši od stanja s slijedi strategiju π. Formalno se za MDP-jeve v_π može definirati sa

$$v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s] = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s],$$

za sve $s \in \mathcal{S}$. Funkciju v_π se naziva *vrijednosna funkcija stanja* za *strategiju* π.

[4] tj. za svaku se akciju dobije kolika je vjerojatnost da će biti odabrana.

Slično, ako se u stanju s odabere akcija a slijedeći strategiju π definira se vrijednost

$$q_\pi(s,a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = s] = \mathbb{E}_\pi[\sum_{k=0}^{\infty}\gamma^k R_{t+k+1}|S_t = s, A_t = a]$$

Funkcija q_π zove se *vrijednosna funkcija akcije* za *strategiju* π.

Vrijednosne funkcije v_π i q_π mogu se procijeniti iz iskustva - ako agent slijedi strategiju π i za svako stanje prati ukupnu dobit koja slijedi nakon tog stanja, onda ti prosjeci konvergiraju k vrijednosti toga stanja $v_\pi(s)$ kada broj posjeta tom stanju teži prema beskonačno. Ako se uz to odvojeno prati prosjeke dobiti za svaku akciju poduzetu iz svakog stanja na analogan način procjenjuje se $q_\pi(s,a)$. Ovakve tehnike procjene nazivaju se *Monte Carlo metode* jer se temelje na prosjeku mnogih slučajnih uzoraka stvarnih dobiti.

Iz otprije navedene definicije *očekivane dobiti* može se primijetiti da ju je moguće izraziti preko rekurzivne relacije

$$\begin{aligned} G_t &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + ... \\ &= R_{t+1} + \gamma[R_{t+2} + \gamma^1 R_{t+3} + \gamma^2 R_{t+4} + ...] \\ &= R_{t+1} + \gamma G_{t+1} \end{aligned} \tag{1.2}$$

Slično vrijedi i za vrijednosne funkcije. Za proizvoljnu strategiju π i stanje s vrijedi

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi[G_t|S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1}|S_t = s] \\ &= \sum_a \pi(a|s) \sum_{s'} \sum_r \mathcal{P}_0(s', r|s,a)[r + \gamma \mathbb{E}_\pi[G_{t+1}|S_{t+1} = s']] \\ &= \sum_a \pi(a|s) \sum_{s',r} \mathcal{P}_0(s', r|s,a)[r + \gamma v_\pi(s')], \forall s \in \mathcal{S} \end{aligned} \tag{1.3}$$

Jednadžbu (1.3) nazivamo *Bellmanova jednadžba* za v_π - ona izražava vezu između vrijednosti trenutnog stanja i vrijednosti stanja koje slijedi nakon njega. Bellmanova jednadžba uzima u obzir sva stanja koja okolina može proizvesti, pridajući svakom vjerojatnost hoće li se dogoditi. Lako se pokaže da je vrijednosna funkcija v_π jedinstveno rješenje pripadne Bellmanove jednadžbe. Kroz tri sljedeća poglavlja koristit će se Bellmanove jednadžbe kao temelj za izračun, odnosno procjenu, v_π.

1.4 Primjer: Mrežni svijet

Slika 1.2: *Mrežni svijet: dinamika nagrada (lijevo) i vrijednosna funkcija stanja (desno) pri korištenju strategije s jednakom vjerojatnošću kretanja u sva četiri smijera* [1]

Slika (1.2) predstavlja model jednostavnog konačnog MDP-a. Svako polje na ploči predstavlja jedno od stanja okoline. Na svakom polju agent može birati između četiri akcije: gore, dolje, lijevo i desno. Svaka akcija deterministički pomiče agenta u tom smjeru na ploči. Akcija koja bi pomakla agenta sa ploče, prelazak ruba ploče, ne mijenja položaj agenta, ali nosi sa sobom nagradu -1. Sve druge akcije donose nagradu 0, osim akcija koje se poduzmu u posebnim stanjima A i B. U stanju A sve četiri akcije donose agentu nagradu +10 i prebacuju ga u stanje A'. U stanju B sve četiri akcije donose agentu nagradu +5 i prebacuju ga u stanje B'.

Uz pretpostavku da agent svaku od akcija bira s jednakom vjerojatnošću na slici (1.2) s desne strane mogu se vidjeti vrijednosti stanja za danu strategiju s korekcijom $\gamma = 0.9$. Vrijednosna funkcija je izračunata rješavanjem sustava jednadžbi (1.3) i lako se provjeri vrijede li, npr. za vrijednost 0.7 u samom središtu tablice vrijedi

$$\begin{aligned} 1/4 * (0 + 0.9 * 2.3) + 1/4 * (0 + 0.9 * 0.4) + \\ 1/4 * (0 - 0.9 * 0.4) + 1/4 * (0 + 0.9 * 0.7) = \\ 1/4 * 0.9[2.3 + 0.4 - 0.4 + 0.7] = \\ 0.67499 \approx 0.7 \end{aligned}$$

1/4 je vjerojatnost odabira svake akcije (pomoću nje se izračunava očekivanje), 0 u svakoj od zagrada je izravna nagrada za odabir te akcije, 0.9 korekcija svake vrijednosti idućeg stanja i na kraju se vrijednost zaokružuje na jednu decimalu.

Negativne vrijednosti stanja na donjem rubu su posljedica velike vjerojatnosti da će se prijeći rub ako se biraju podjednako sve akcije. Stanje A je najbolje stanje koju god strategiju odabrali - ali je njegova vrijednost manja od nagrade koja se sigurno dobije u njemu jer se iz stanja A prelazi direktno u stanje A' u kojem je visoka vjerojatnost da će agent prijeći rub ploče. S druge strane, vrijednost stanja B je veća nego nagrada koja se u njemu dobije jer ono vodi u stanje B' čija je vrijednost također pozitivna - vjerojatnost kazne (negativna nagrada) za prelazak granice iz B' niža je od vjerojatnosti da će agent opet nabasati na stanja A ili B.

U ovom su primjeru dani pozitivni skalari kao nagrade za postizanje cilja, odnosno negativni skalari za prelazak ruba kao svojevrsna kazna i nula u svim ostalim slučajevima. Međutim, je li bitan predznak nagrada koje dajemo ili samo razlika između njih? Ako se na svaku nagradu u ovom primjeru doda neka konstanta c uvrštavanjem u (1.1) dobije se:

$$G_t = \sum_{k=0}^{\infty} \gamma^k [R_{t+k+1} + c] = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} + c \sum_{k=0}^{\infty} \gamma^k \tag{1.4}$$

Kako se u pravilu uzima γ manja od 1, to je desni pribrojnik konvergentni geometrijski red. Dakle dodavanje konstante c svakoj nagradi pomiče vrijednost svakog stanja za $v_c = c \sum_{k=0}^{\infty} \gamma^k$ što ne utječe na odnos vrijednosti stanja - ovo vrijedi za sve strategije.

1.4.1 Optimalna strategija

Riješiti problem PU, ugrubo govoreći, znači pronaći strategiju koja dugoročno prikupi puno nagrada - što nasumično pogađanje i pretpostavljanje koje se do sada koristilo definitivno nije postiglo. U slučaju konačnih MDP-ova optimalnu strategiju može se i strogo definirati. Prvo na skupu svih strategija definira se uređaj uz pomoć njihovih vrijednosnih funkcija.

Definicija 1.2 *Neka su za MDP $\mathcal{M}$ sa skupom stanja $\mathcal{S}$ dane strategije π i π' sa vrijednosnim funkcijama v_π i $v_{\pi'}$ redom. Za strategiju π kažemo da je bolja ili jednaka strategiji π', i pišemo $\pi \geq \pi'$, ako za svaki $s \in \mathcal{S}$ vrijedi $v_\pi(s) \geq v_{\pi'}(s)$.*

Važno je uočiti da uvijek postoji barem jedna strategija koja je bolja ili jednaka od svih ostalih[5]. Iako možda postoji više takvih strategija sve ih se označava sa π_* i nazivaju se *optimalna strategija*.

[5]Za svako stanje $s \in \mathcal{S}$ postoji neka strategija π_s za koju je vrijednost stanja s najveća - tj. veća ili jednaka od vrijednosti tog stanja za bilo koju drugu strategiju. Sada se jednostavno konstruira nova strategija π_* koja će se u svakom stanju $s \in \mathcal{S}$ ponašati kao strategija π_s. Lako se vidi da je strategija π_* bolja ili jednaka od svih drugih.

Sve optimalne strategije imaju istu vrijednosnu funkciju čiji je naziv *optimalna funkcija stanja* i definira se sa

$$v_*(s) = \max_\pi v_\pi(s)$$

za sve $s \in \mathcal{S}$.

Uz to sve optimalne strategije imaju i zajedničku *optimalnu vrijednosnu funkciju akcije* definiranu sa

$$q_*(s,a) = \max_\pi q_\pi(s,a)$$

za sve $s \in \mathcal{S}$ i sve $s \in \mathcal{A}$.

U kontekstu funkcije v_* lako se uočiti da vrijedi

$$q_*(s,a) = \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) | S_t = s, A_t = a]$$

.

Jer je v_* vrijednosna funkcija za neku strategiju, ona mora zadovoljavati uvjete dane Bellmanovim jednadžbama (1.3). Ali budući da je v_* optimalna vrijednosna funkcija ti se uvjeti mogu zapisati na način da se ne referira na iti jednu specifičnu strategiju - ovaj se zapis zove *Bellmanova jednadžba optimalnosti*:

$$\begin{aligned}
v_*(s) &= \max_{a \in \mathcal{A}(s)} q_{\pi*}(s,a) && (1.5)\\
&= \max_a \mathbb{E}_{\pi*}[G_t | S_t = s, A_t = a] \\
&= \max_a \mathbb{E}_{\pi*}[R_{t+1} + \gamma G_{t+1} | S_t = s, A_t = a] \\
&= \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) | S_t = s, A_t = a] && (1.6)\\
&= \max_a \sum_{s',r} \mathcal{P}_0(s', r | s, a)[r + \gamma v_*(s')], && (1.7)
\end{aligned}$$

za sve $s \in \mathcal{S}$

Bellmanova jednadžba optimalnosti na intuitivnoj razini govori da vrijednost stanja u optimalnoj strategiji mora biti jednaka očekivanoj dobiti za odabir najbolje akcije u tom stanju

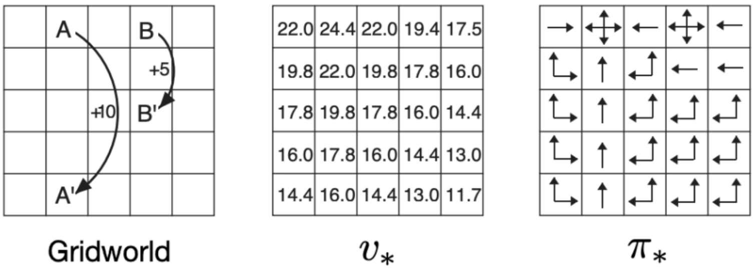
(1.6) i (1.7) su dva oblika Bellmanove jednadžbe optimalnosti za v_*. Za q_* Bellmanova jednadžba optimalnosti je

$$\begin{aligned} q_*(s,a) &= \mathbb{E}[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') | S_t = s, A_t = a] \\ &= \sum_{s',r} \mathcal{P}_0(s', r | s, a)[r + \gamma \max_{a'} q_*(s', a')] \end{aligned} \tag{1.8}$$

Za konačne MDP-ove Bellmanova jednadžba optimalnosti (1.7) za v_* ima jedinstveno rješenje - radi se zapravo o sustavu jednadžbi, po jedna za svako stanje. Ako je poznata dinamika okoliša, tj. matrica prijelaza $\mathcal{P}_0$, sustav je općenito rješiv bilo kojom metodom za rješavanje sustava nelinearnih jednadžbi. Kada se radi izračun za v_*, lako je odrediti optimalnu strategiju - za svako stanje s postojat će barem jedna akcija koja daje maksimum postignut u Bellmanovoj jednadžbi optimalnosti. Bilo koja strategija koja sa vjerojatnošću većom od nula bira te i samo te akcije bit će optimalna. Analogno zaključivanje bi bilo da se radi o *pohlepnom*[6] ponašanju s obzirom na optimalnu funkciju vrijednosti v_*.

Kada se ovo primijeni na spomenuti primjer Mrežnog svijeta rješavanjem Bellmanove jednadžbe za v_*: može se vidjeti na slici (1.3), u sredini su prikazane optimalne vrijednosti za svako stanje te desno optimalna strategija za ovaj primjer.

[6] Pohlepno donošenje odluka bi značilo biranje opcija uzimajući u obzir samo lokalna tj. trenutna saznanja bez obzira na dalekosežne posljedice - općenito znači odabir **trenutno** najboljih opcija. Ljepota v_* leži u tome da kada se koristi za određivanje trenutnih posljedica akcija, pohlepna je strategija zapravo optimalna jer v_* sadrži u sebi i sve moguće buduće nagrade.



Slika 1.3: Mrežni svijet: optimalna rješenja [1]

Iako rješavanje Bellmanove optimalne jednadžbe daje optimalnu strategiju za problem PU-a, ova metoda je sama po sebi rijetko kad korisna. Da bi ju se uopće razmotrilo trebaju biti ispunjena tri uvjeta : (1) potpuno poznavanje okoline, (2) dovoljan broj računalnih resursa za izračun; (3) sva stanja imaju Markovljevo svojstvo;

U stvarnom svijetu rijetko je slučaj da su sva tri uvjeta ispunjena, a najčešće nije nijedan. Npr. za partiju igre Backgammon, isto analogno vrijedi i za Go ili Šah, iako vrijede uvjeti (1) i (3), uvjet (2) je nepremostiva prepreka: igra ima 10^{20} različitih stanja i najmoćnijim današnjim računalima bi trebale tisuće godina za rješavanje pripadnih Bellmanovih jednadžbi.

U podržanom učenju je stoga potrebno zadovoljiti se približnim rješenjima čime će se pozabaviti poglavlja koja slijede.

Poglavlje 2

Dinamičko programiranje

Dinamičko programiranje (DP) se odnosi na niz algoritama koji se koriste za izračun optimalne strategije u slučaju da postoji savršen model okoliša kao MDP-a. Iako od klasičnih algoritama DP-a nema velike koristi u praksi - zbog zahtjeva za savršenim modelom okoline ili prevelikih računalnih zahtjeva - ipak su važni iz teoretske perspektive jer sve ostale metode, koje će biti naknadno spomenute, u nekoj se mjeri temelje na DP-u: pokušavaju doći do istih rezultata kao DP samo s manje računanja ili bez savršenog modela okoline. U ovom poglavlju, a i nadalje, pretpostavka je da se radi s konačnim MDP-om, tj. da su svi skupovi MDP-a konačni.

Općenito govoreći, osnovna je ideja DP-a, i PU-a korištenje vrijednosnih funkcija kako bi se olakšala potraga za optimalnom strategijom. Fokus ovog poglavlja je prikaz načina korištenja DP-a za izračun vrijednosnih funkcija definiranih u prethodnom poglavlju. Već je otprije poznato, kada su jednom utvrđene optimalne vrijednosne funkcije v_* ili q_* lako se dolazi do optimalne strategije. Da bi se došlo do optimalnih vrijednosnih funkcija DP se koristi Bellmanovim optimalnim jednadžbama (1.5) i (1.8) kao pravilom ažuriranja kojim poboljšava aproksimacije funkcije koju računa.

2.1 Vrednovanje strategije (predviđanje)

Prvo je vrijedno razmotriti kako izračunati vrijednosnu funkciju stanja v_π za proizvoljnu strategiju π. U literaturi se ovo zove *vrednovanje strategije* ili *problem predviđanja.* U prošlom je poglavlju pokazano za sve $s \in \mathcal{S}$

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi[G_t|S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1}|S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s] \\ &= \sum_a \pi(a|s) \sum_{s',r} \mathcal{P}_0(s', r|s, a)[r + \gamma v_\pi(s')], \end{aligned} \tag{2.1}$$

gdje je $\pi(a|s)$ vjerojatnost da će agent poduzeti akciju a ako se nalazi u stanju s i slijedi strategiju π. Očekivanja imaju index π da bi naznačili uvjetovanost o π - tj. da vrijedi ako se slijedi strategiju π.

Funkcija v_π postoji i jedinstvena je ako je $\gamma < 1$ ili agent sigurno dolazi do terminalnog stanja - do eventualnog cilja ili kraja istraživanja - odakle god počeli, slijedeći strategiju π.

Ako je dinamika okoliša u potpunosti poznata , onda vrijedi (2.2) sustav od $|\mathcal{S}|$ jednadžbi s $|\mathcal{S}|$ nepoznanica. Iako rješavanje sustava može biti računalno zahtjevno, ono je općenito dosta jednostavno s teoretske strane. U ovom su slučaju najprikladnije iterativne metode: Promatra se niz aproksimacija vrijednosnih funkcija $v_0, v_1, v_2, ...$, gdje svaka ide iz $\mathcal{S}$ u $\mathbf{R}$. Prva se aproksimacija uzima proizvoljno (uz pretpostavku da je vrijednost terminalnih stanja, ako ih ima, 0), a sve sljedeće se dobiju koristeći Bellmanove jednadžbe za v_π (2.2) kao pravilo ažuriranja:

$$
\begin{aligned}
v_{k+1}(s) &= \mathbb{E}_{\pi}[R_{t+1} + \gamma v_k(S_{t+1}) | S_t = s] \\
&= \sum_a \pi(a|s) \sum_{s',r} \mathcal{P}_0(s', r|s, a)[r + \gamma v_k(s')],
\end{aligned} \tag{2.2}
$$

, za sve $s \in \mathcal{S}$.

Očito je $v_k = v_\pi$ fiksna točka ovog pravila ažuriranja jer Bellmanove jednadžbe za v_π osiguravaju jednakost u tom slučaju. Općenito, može se pokazati da niz v_k konvergira ka v_π kada $k \to \infty$ pod istim uvjetima koji garantiraju egzistenciju v_π. Ovaj se algoritam naziva *iterativno vrednovanje strategije.*

Algoritam 1: *Iterativno vrednovanje strategije*

Podatci: strategija π koju treba vrednovati

Odredi: mali $\epsilon > 0$ kojim određujemo željena preciznost

Za sve $s \in \mathcal{S}$ proizvoljno postavi $V(s)$ uz uvjet da je $V(terminal) = 0$;

$\Delta \leftarrow \epsilon + 1$;

dok $\Delta > \epsilon$ **čini**

  $\Delta \leftarrow 0$;

  za $s \in \mathcal{S}$ **čini**

    $v \leftarrow V(s)$;

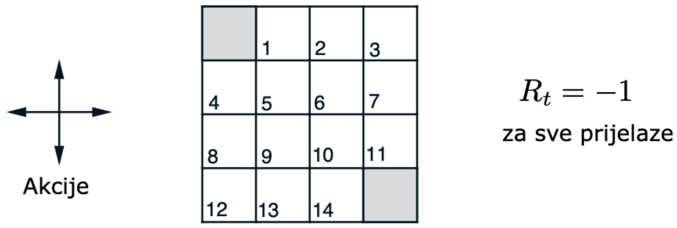
    $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} \mathcal{P}_0(s', r|s, a)[r + \gamma V(s')]$;

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

  kraj

kraj

Primjer 2.1 (Mrežni svijet: 4x4) *U mrežnom svijetu sa slike ([1]) vrijedi: Neterminalna stanja su* $\mathcal{S} = 1, 2, ..., 14$, *za svako stanje moguće su 4*

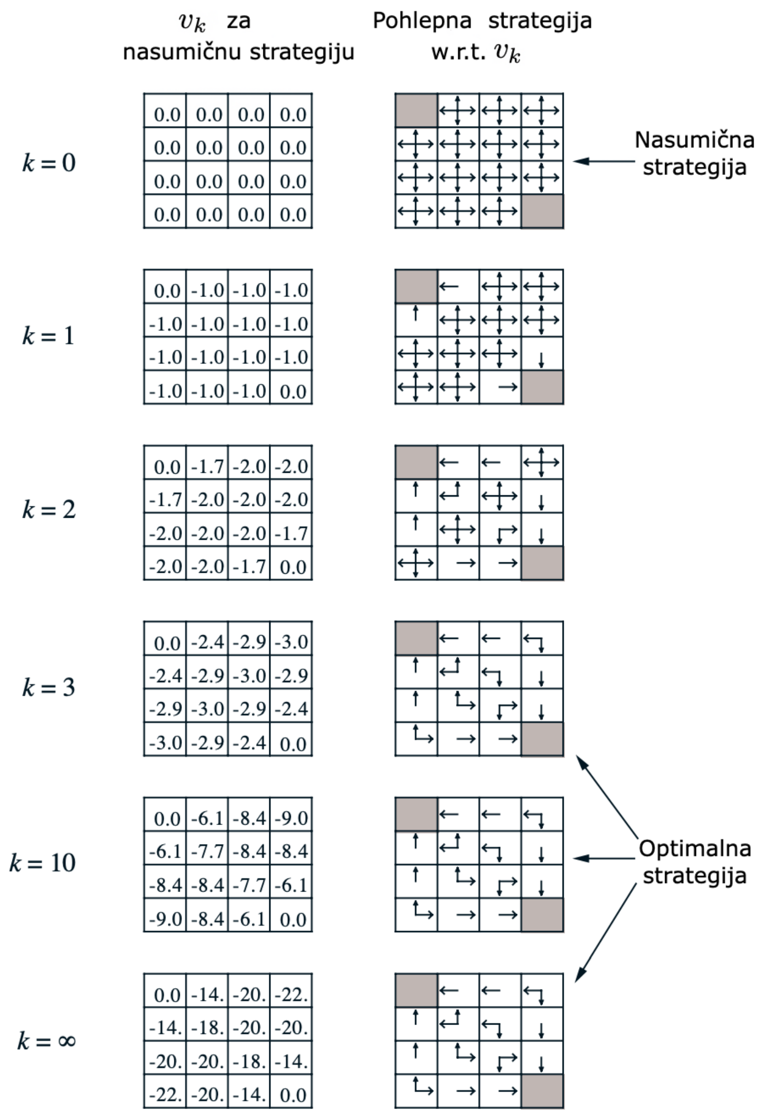


akcije $\mathcal{A} = \{gore, dolje, lijevo, desno\}$ *koje deterministički prebacuju agenta u pripadno stanje, osim ako bi prešao rub ploče, u tom slučaju stanje ostaje nepromijenjeno.*

Nagrada iznosi -1 za sve prijelaze dok agent ne dođe do terminalnog stanja (siva polja na ploči).

Dakle, vrijedi npr. $\mathcal{P}_0(6, -1|5, desno) = 1$, $\mathcal{P}_0(7, -1|7, desno) = 1$ *i* $\mathcal{P}_0(10, r|5, desno) = 0$ *za sve* $r \in \mathcal{R} \subset \mathbf{R}$.

Na lijevoj strani slike (2.1), na stranici 18, vidi se niz vrijednosnih funkcija $\{v_k\}$ *dobivenih iterativnim vrednovanjem nasumične strategije - agent s jednakom vjerojatnosti, 0.25, bira svaku od akcija.*



Slika 2.1: Konvergencija iterativnog vrednovanja strategije. Lijevi stupac prikazuje niz aproksimacija vrijednosne funkcije stanja za jednakovjerojatnu nasumičnu strategiju. U desnom stupcu vidi se niz pohlepnih strategija temeljenih na izračunatim procjenama vrijednosnih funkcija. Strategija dobivena na kraju je garantirano bolja od nasumične, odnosno one na kojoj su temeljene procjene funkcija, ne nužno i općenito optimalna - ali u ovom slučaju je. [1]

2.2 Poboljšavanje strategije (kontrola)

Razlog zašto uopće računati vrijednosnu funkciju neke strategije je da bi se pomoću nje mogle pronaći bolje strategije, tzv. *kontrola.* Uz pretpostavka da je izračunata vrijednosna funkcija v_π neke determinističke strategije π, za neko stanje s vrijedi znati isplati li se promijeniti strategiju da se u tom stanju izabere akciju $a \neq \pi(s)$. Ono što je moguće učiniti da bi se odgovorilo na ovo pitanje je izabirati akciju a u stanju s, a u svim ostalim slučajevima slijediti strategiju π. Vrijednost ovakvog načina ponašanja je

$$q_\pi(s,a) = \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s, A_t = a] = \sum_{s',r} \mathcal{P}_0(s', r|s,a)[r + \gamma v_\pi(s')],$$

Pitanje je je li ova vrijednost veća ili manja od $v_\pi(s)$. U slučaju da je veća, može se zaključiti da je novonastala strategija bolja od prijašnje. Točnost takvog zaključivanja je posljedica općenitijeg rezultata poznatog kao *Teorem poboljšanja strategije.*

Teorem 2.2 (poboljšanja strategije) *Neka su π i π' strategije takve da za njih vrijedi*

$$\forall s \in \mathcal{S}, q_\pi(s, \pi'(s)) \geq v_\pi(s).$$

Tada je strategija π' barem jednako dobra kao strategija π, tj. za svako stanje $s \in \mathcal{S}$ vrijedi

$$v_{\pi'}(s) \geq v_\pi(s).$$

Dokaz. Neka vrijedi (2.2), tj $q_\pi(s, \pi'(s)) \geq v_\pi(s)$, onda vrijedi i sljedeće

$$
\begin{aligned}
v_\pi(s) &\leq q_\pi(s, \pi'(s)) && \text{(primijeni (2.2))} \\
&= \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s, A_t = \pi'(s)] \\
&= \mathbb{E}_{\pi'}[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s] \\
&\leq \mathbb{E}_{\pi'}[R_{t+1} + \gamma q_\pi(S_{t+1}, \pi'(S_{t+1})) | S_t = s] \\
&= \mathbb{E}_{\pi'}[R_{t+1} + \gamma \mathbb{E}[R_{t+2} + \gamma v_\pi(S_{t+2}) | S_{t+1}, A_{t+1} = \pi'(S_{t+1})] | S_t = s] \\
&= \mathbb{E}_{\pi'}[R_{t+1} + \gamma R_{t+2} + \gamma^2 v_\pi(S_{t+2}) | S_t = s] \\
&\leq \mathbb{E}_{\pi'}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 v_\pi(S_{t+3}) | S_t = s] \\
&\vdots \\
&\leq \mathbb{E}_{\pi'}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots | S_t = s] \\
&= v_{\pi'}(s).
\end{aligned}
$$

■

Sada se primjenjuje ista logika na *sva* stanja: u svakom stanju agent izabire akciju koja je najbolja po $q_\pi(s,a)$ Tj. formira se novu *pohlepnu* strategiju π' definiranu s

$$
\begin{aligned}
\pi'(s) &= \operatorname*{argmax}_a q_\pi(s,a) \\
&= \operatorname*{argmax}_a \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s, A_t = a] \\
&= \operatorname*{argmax}_a \sum_{s',r} \mathcal{P}_0(s', r | s, a)[r + \gamma v_\pi(s')]
\end{aligned} \tag{2.3}
$$

gdje argmax_a označava vrijednost za koju je izraz maksimalan. Konstrukcija ovakve pohlepne strategije zadovoljava uvjete Teorema (2.2) tako da je jasno da je ova strategija barem jednako dobra kao početna strategija - možda čak i bolja.

Očigledno, ako vrijedi pretpostavka da je nova pohlepna strategija π' jednako dobra kao i izvorna π i da **nije** bolja od nje, onda vrijedi $v_\pi = v_{\pi'}$ te iz (2.3) slijedi:

$$\begin{aligned} v_{\pi'}(s) &= \max_a \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s, A_t = a] \\ &= \max_a \sum_{s',r} \mathcal{P}_0(s', r|s, a)[r + \gamma v_\pi(s')], \forall s \in \mathcal{S}. \end{aligned} \tag{2.4}$$

Budući da je ovo isto što i Bellmanova jednadžba optimalnosti (1.5), onda $v_{\pi'}$ mora biti i v_*. Pa su i π i π' optimalne strategije. Proces poboljšanja strategije će uvijek dati bolju strategiju od polazne, osim ako je polazna strategija već bila optimalna[1].

2.3 Iteracija strategija

U ovoj se fazi iskoristio v_π da bi se poboljšalo strategiju π i došlo do strategije π', pa se može nastaviti analogno, tj. izračunati vrijednosnu funkciju $v_{\pi'}$ i iskoristiti je da bi došli do još bolje strategije π''. Na ovaj način dobije se niz monotono rastućih strategija (s obzirom na uređaj koji je definiran) i njima pripadajućih vrijednosnih funkcija

$$\pi_0 \xrightarrow{\mathbb{E}} v_{\pi_0} \xrightarrow{\mathbb{I}} \pi_1 \xrightarrow{\mathbb{E}} v_{\pi_1} \xrightarrow{\mathbb{I}} \pi_2 \xrightarrow{\mathbb{E}} \cdots \xrightarrow{\mathbb{I}} \pi_* \xrightarrow{\mathbb{E}} v_{\pi_*}$$

gdje $\xrightarrow{\mathbb{E}}$ predstavlja *vrednovanje* strategije, a $\xrightarrow{\mathbb{I}}$ *poboljšanje* strategije - eng. *improvement*. Svaka iduća strategija je izričito bolja od prethodne (osim ako je ova već bila optimalna). Jer konačan MDP ima konačno mnogo determinističkih strategija, to ovaj proces sigurno konvergira ka optimalnoj strategiji i optimalnoj funkciji vrijednosti u konačno mnogo iteracija. Ovakav način pronalaska optimalne strategije naziva se *iteracija strategija*.

[1]Slično zaključivanje i dokazi se mogu primijeniti i za stohastičke strategije.

Algoritam 2: *Iteracija strategija za procjenu* $\pi \approx \pi_*$

Inicijaliziraj: za sve $s \in \mathcal{S}$ (nasumično) postavi $V(s) \in \mathbf{R}$ i $\pi(s) \in \mathcal{A}(s)$, $V(terminal) = 0$

Vrednovanje strategije:

$\Delta \leftarrow \epsilon + 1$;

dok $\Delta > \epsilon$ **čini**

  $\Delta \leftarrow 0$;

  za $s \in \mathcal{S}$ **čini**

    $v \leftarrow V(s)$;

    $V(s) \leftarrow \sum_{s',r} \mathcal{P}_0(s', r|s, \pi(s))[r + \gamma V(s')]$;

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

  kraj

kraj

Poboljšanje strategije:

strategija-stabilna $\leftarrow true$;

za $s \in \mathcal{S}$ **čini**

  stara-akcija $\leftarrow \pi(s)$;

  $\pi(s) \leftarrow \text{argmax}_a \sum_{s',r} \mathcal{P}_0(s', r|s, a)[r + \gamma V(s')]$;

  ako *stara-akcija* $\neq \pi(s)$ **onda**

    strategija-stabilna $\leftarrow false$;

  kraj

kraj

ako *strategija-stabilna* **onda**

  zaustavi algoritam i vrati $V \approx v_*$ i $\pi \approx \pi_*$;

kraj

inače

  idi na *Vrednovanje strategije*;

kraj

2.4 Iteracija vrijednosti

Negativna strana iteracije strategije je to što svaka iteracija uključuje vrednovanje strategije - što može biti zahtjevan račun. Ako se vrednovanje strategije radi iterativno, onda se konvergencija ka optimalnoj funkciji vrijednosti v_* postiže tek na kraju, tj. u graničnoj vrijednosti. U primjeru na slici (2.1) vidljivo je da svako vrednovanje strategije nakon treće iteracije nema nikakav utjecaj na dobivenu pohlepnu strategiju - ovo sugerira da bi se neka vrednovanja strategija moglo preskočiti.

Štoviše, pokaže se da postoji nekoliko kriterija po kojima je moguće preskočiti vrednovanje strategije uz zadržavanje garantirane konvergencije ka optimalnoj strategiji.

Posebno je zanimljiv slučaj kada vrednovanje strategije staje nakon samo jednog prolaza po svim stanjima - ovaj se algoritam zove *iteracija vrijednosti*. Opisan je jednostavnim pravilom ažuriranja koje kombinira poboljšanje strategije i preskakanje koraka vrednovanja strategije:

$$\begin{aligned} v_{k+1}(s) &= \max_a \mathbb{E}[R_{t+1} + \gamma v_k(S_{t+1}) | S_t = s, A_t = a] \\ &= \max_a \sum_{s',r} \mathcal{P}_0(s', r|s, a)[r + \gamma v_k(s')], \forall s \in \mathcal{S}. \end{aligned} \tag{2.5}$$

Za proizvoljno odabrani v_0 niz $\{v_k\}$ konvergira ka v_* po istim uvjetima koji garantiraju postojanje v_*. Algoritam se izvodi iz Bellmanovih jednadžbi (1.5) - iteracija se dobije tako što se Bellmanova jednadžba pretvori u pravilo ažuriranja.

Vrijedi spomenuti i način zaustavljanja algoritma. Kao i kod vrednovanja strategije, iteracije vrijednosti formalno trebaju beskonačno mnogo iteracija da bi konvergirale ka v_*. U praksi se zaustavljaju kada se vrijednosna funkcija u jednom prijelazu promijeni za dovoljno malu vrijednost. Precizniji zapisi su vidljivi u Algoritmu (3).

Algoritam 3: *Iteracija vrijednosti za procjenu* $\pi \approx \pi_*$

Inicijaliziraj: za sve $s \in \mathcal{S}$ (nasumično) postavi $V(s) \in \mathbf{R}$,
$V(terminal) = 0$ te odaberi mali $\epsilon > 0$ kojim se
određuje željenu preciznost

$\Delta \leftarrow \epsilon + 1$;

dok $\Delta > \epsilon$ **čini**

  $\Delta \leftarrow 0$;

  za $s \in \mathcal{S}$ **čini**

    $v \leftarrow V(s)$;

    $V(s) \leftarrow \max_a \sum_{s',r} \mathcal{P}_0(s', r|s, a)[r + \gamma V(s')]$;

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

  kraj

kraj

Vrati determinističku strategiju $\pi \approx \pi_*$ za koju je

$\pi(s) = \mathrm{argmax}_a \sum_{s',r} \mathcal{P}_0(s', r|s, a)[r + \gamma V(s')]$;

Iteracija vrijednosti zapravo kombinira, u svakom prolazu kroz stanja, prolaz vrednovanja strategije i prolaz poboljšanja strategije.

2.5 Napomene

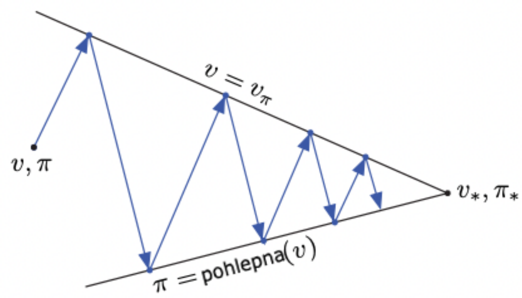
2.5.1 Asinkrono DP

U slučaju da je skup stanja jako velik čak i jedan prolaz skupom može bit računalno jako "skup." Asinkroni DP algoritmi ne ovise o sistematskom ažuriranju **svih** stanja - ažuriraju se stanja proizvoljnim redom, koristeći pri tome dostupne vrijednosti stanja. Može se dogoditi da je vrijednost nekog stanja ažurirana više puta dok nekog drugog nije niti jednom. Međutim, izbjegavanje punog prolaza po stanjima ne znači da će se nužno proći jeftinije jer sve vrijednosti stanja treba prije ili kasnije ažurirati - ne mogu se neke jednostavno ignorirati - ali nije potrebno čekati da se izvrte sve vrijednosti da bi mogli unaprijediti tu strategiju.

2.5.2 Generalizirana iteracija strategije

Iteracija strategije se sastoji od dva isprepletena procesa: izračun vrijednosne funkcije da odgovara trenutnoj strategiji (vrednovanje strategije) i formiranje pohlepne strategije s obzirom na trenutnu vrijednosnu funkciju (poboljšanje strategije). Ovi procesi alterniraju, da bi idući mogao započeti prethodni mora završiti - što zapravo i nije nužno. Kod asinkronih DP metoda ovi su procesi isprepleteni na finiji način i dokle god proces ažurira sva stanja konačni rezultati bi trebali biti isti - konvergencija ka optimalnoj vrijednosnoj funkciji i optimalnoj strategiji.

Pod nazivom *generalizirana iteracija strategija* (GPI) misli se na općenitu ideju međudjelovanja *vrednovanja strategije* i *poboljšanja strategije* neovisno o granularnosti i detaljima tih procesa. Skoro sve metode PU-a može se opisati kao GPI - sve imaju strategije i vrijednosne funkcije, s tim da se strategije poboljšavaju s obzirom na vrijednosne funkcije, a funkcije se računaju po strategiji.



Slika 2.2: GPI [2]

Jednom kad se oba procesa stabiliziraju, tj. nema više promjena, znači da je postignuta optimalna vrijednosna funkcija i optimalna strategija - a to se događa samo ako je vrijednosna funkcija konzistentna s trenutnom strategijom i trenutna je strategija pohlepna s obzirom na vrijednosnu funkciju.

Odnos vrednovanja i poboljšanje u GPI-u može se promatrati kao suparništvo i suradnja. Suparništvo jer svaki proces na neki način povlači na svoju stranu - formiranje pohlepne strategije s obzirom na trenutnu funkciju vrijednosti čini funkciju netočnom za novu strategiju, a izračun nove funkcije čini da strategija nije više pohlepna u odnosu na nju. Međutim, dugoročno ova dva procesa dolaze do jednog zajedničkog rješenja: optimalne vrijednosne funkcije i optimalne strategije.

Poglavlje 3

Predviđanje bez modela

Metode koje su dosad spomenute su jako korisne i vrijedne, no imaju jako veliki nedostatak koji je bio prisutan iako ne i istaknut - ove se metode mogu koristiti samo ako je dinamika okoliša u potpunosti poznata. Ako je smjer iz kojeg će vjetar zapuhati nepoznat, ili nisu poznati svi parametri koji utječu na produktivnost tvornice, onda se ne mogu ni koristiti. Upravo su zato korisne *metode predviđanja bez modela*, eng. model-free prediction, za procjenu vrijednosne funkcije i otkrivanje optimalne strategije. Postoje dvije glavne klase metoda predviđanja bez modela:

Monte-Carlo učenje - metode koji prijeđu cijelu putanju agenta i procjenjuju vrijednost iz dobiti uzoraka;

Učenje s vremenskom razlikom, eng. *temporal-difference learning* - metode koje gledaju jedan korak unaprijed i procjenjuju dobit nakon tog jednog koraka.

Dakle, odbacuje se ogromnu pretpostavku ponašanja okoliša - koja je nerealna za većinu zanimljivih problema - i radi se s metodama koje koriste *iskustvo* agenta, tj. njegove interakcije s okolišem. Kao i do sada, metoda traženja optimalne strategije podijeljena je na dva dijela: vrednovanje strategije, tj. nalaženje vrijednosne funkcije za danu strategiju, i poboljšanje dane strategije. Predviđanje se bavi prvim dijelom - ako je zadana strategija, vrši se procjena koliko je dobra, tj. traži se njenu vrijednosnu funkciju.

3.1 Monte-Carlo Učenje

Pojam "Monte-Carlo" se općenito odnosi na metode procjene koje u sebi sadrže neki nasumični dio koji ih u značajnoj mjeri određuje. Ovdje se taj pojam koristi za metode koje računaju prosjeke cjelokupne dobiti - za razliku od metoda koje promatraju djelomičnu dobit kojima se bavi iduće poglavlje.

MC koristi najjednostavniju ideju za slučaj da modela nema: da bi odredio vrijednost nekog stanja računa prosjek dobiti postignutih iz tog stanja. Ako su dana primjerice tri uzorka koja počinju iz istog stanja sa dobitima 7,3 i 8, prosjek tih vrijednosti je 6 pa je pretpostavka da je vrijednost toga stanja 6. Važno je uočiti da epizoda treba završiti kako bi se mogla izračunati dobit, tj. agent treba doći do terminalnog stanja. Kako broj uzoraka raste, aproksimacija teži stvarnoj vrijednosti tog stanja.

Cilj je procjeniti $v_\pi(s)$, vrijednost stanja s ako se slijedi strategiju π, a poznat je skup epizoda u kojima je agent slijedio strategiju π i prošao kroz stanje s. Svako pojavljivanje stanja s u epizodi naziva se *posjet* stanju s. Kako agent u jednoj epizodi može posjetiti isto stanje više puta:

MC metoda prvog posjeta procjenjuje vrijednost $v_\pi(s)$ kao prosjek[1] dobiti od prvog posjeta stanju s, dok *MC metoda svakog posjeta* računa prosjek

[1] Inkrementalni izračun aritmetičke sredine (vidi algoritam (4)):

Arit. sredine $\mu_1, \mu_2, ...$ niza $x_1, x_2, ...$ može se računati inkrementalno:

$$\mu_k = \frac{1}{k}\sum_{j=1}^{k} x_j = \frac{1}{k}(x_k + \sum_{j=1}^{k-1} x_j) = \frac{1}{k}(x_k + (k-1)\mu_{k-1}) = \mu_{k-1} + \frac{1}{k}(x_k - \mu_{k-1})$$

nakon svakog posjeta stanju s. Iako su ove dvije MC metode jako slične među njima ipak postoje male teoretske razlike.

Algoritam 4: *MC metoda prvog posjeta za procjenu vrijednosti* $V \approx v_\pi$

Zadana je strategija π koju treba procjeniti;

Inicijaliziraj: za sve $s \in \mathcal{S}$ (nasumično) postavi $V(s) \in \mathbf{R}$;

$Dobiti(s) \leftarrow$ prazna lista, za sve $s \in \mathcal{S}$;

dok *true* **čini**

    Generiraj epizodu sljedeći strategiju π:

    $S_0, A_0, R_1, S_1, A_1, R_2, \cdots, S_{T-1}, A_{T-1}, R_T$;

    $G \leftarrow 0$;

    za *svaki korak* $t \in [T-1, T-2, ..., 0]$ **čini**

        $G \leftarrow \gamma G + R_{t+1}$;

        ako *Stanje* S_t *se* ***ne*** *nalazi u nizu* $S_0, S_1, \cdots, S_{t-1}$ **onda**

            Dodaj G na kraj liste $Dobiti(S_t)$;

            $V(S_t) \leftarrow prosjek(Dobiti(S_t))$;

        kraj

    kraj

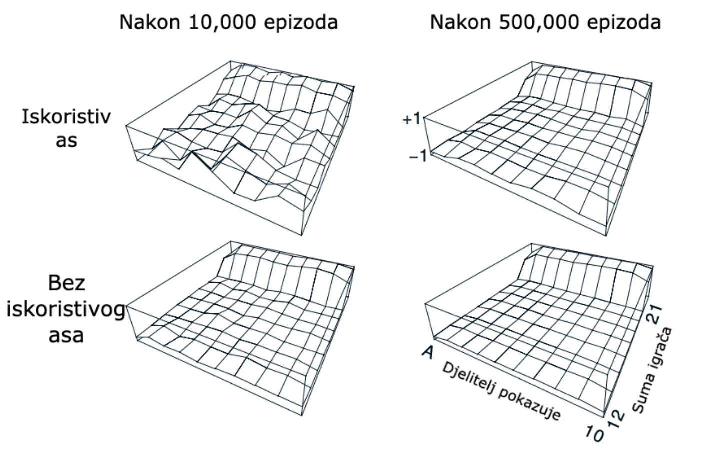
kraj

3.1.1 Primjer: Blackjack

Cilj popularne casino igre je skupiti karte čija je suma što veća a da ne premašuje sumu od 21. Sve karte sa slikom vrijede 10, karte s brojem vrijede koliko pokazuju, osim asa koji vrijedi 1 ili 11 ovisno o tome što više odgovara igraču. Igrač igra protiv djelitelja (kuće, casina). Igra započinje tako što oboje dobiju po dvije karte s tim da je vidljiva samo jedna djeliteljeva karta, dok drugu vidi samo djelitelj. Ako igrač skupi 21 automatski pobjeđuje, osim ako i djelitelj ima 21, u tom slučaju je izjednačeno. Ako je zbroj manji od 21, može se zatražiti još karata jednu po jednu dok ili igrač ne odluči stati ili zbroj na kartama premaši 21. Ako je 21 premašen, igrač gubi tu partiju igre. Ako igrač odluči stati, red je na djelitelju. Djelitelj uvijek igra istu strategiju: vuče karte dok ne postigne sumu 17 ili više. Ako djelitelj prebaci 21, igrač pobjeđuje, inače pobjeđuje onaj koji je bliže 21.

Nagrade su sljedeće: +1, -1 i 0 za pobjedu u igri, poraz i izjednačeno redom. Sve akcije unutar jedne partije daju nagradu 0 i ne radi se korekcija ($\gamma = 0$) tako da nagrade u terminalnom stanju zapravo daju dobit. U svakom potezu, tj. stanju, igrač može birati uzima li još jednu kartu ili staje - stanja ovise o kartama koje ima igrač i karti djelitelja koju svi vidimo. Pretpostavljamo da djelitelj izvlači karte iz beskonačnog špila, tj. da nema smisla brojati karte. Kada igrač ima asa kojeg može brojati kao 11 bez da prijeđe 21, onda se kaže da je taj as *iskoristiv* - u tom slučaju se broji kao 11, kad bi ga brojali kao 1 igrač bi imao manje od 11 pa je poznato koja je njegova iduća akcija. Sigurno će uzeti još jednu kartu jer nema što izgubiti. Znači, igrač donosi odluku temeljenu na tri činjenice: njegova trenutna suma (12-21), karta koju pokazuje djelitelj (as-10) i ima li iskoristivog asa. Sveukupno 200 stanja.

Neka je strategija agenta da stane ako ima sumu 20 ili 21, inače uvijek uzima još jednu kartu. Da bi otkrili vrijednosnu funkciju stanja ove strategije MC metodom simulira se velik broj partija igre koristeći ovu strategiju i računa se prosjek dobiti za svako stanje. Na ovaj način dobije se procjenu vrijednosne funkcije stanja prikazanog na slici (3.1.1) - procjena vrijednosti stanja kada igrač ima iskoristivog asa je manje sigurna jer je to stanje puno rjeđe, ali nakon 500000 partija vrijednosna funkcija je dosta precizno proci-



Slika 3.1: Procjena vrijednosne funkcije stanja MC metodom za strategiju koja staje samo na 20 i 21 [2]

jenjena.

Iako je u ovom slučaju znanje o okolini potpuno, ipak nije jednostavno primijeniti algoritme DP-a na njega. DP metode pretpostavljaju poznatu distribuciju vjerojatnosti za idući događaj, tj. da je poznata matrica prijelaza $\mathcal{P}_0$ - a nju nije lako odrediti za Blackjack. Sve moguće vjerojatnosti događaja bi trebalo izračunati prije nego li se primijene algoritmi DP-a, a ti izračuni su često komplicirani i podložni greškama. S druge strane, generiranje partija potrebnih za MC metode je iznimno lako i baš je ova činjenica velika prednost MC metode čak i ako je dinamika okoline u potpunosti poznata.

Dodatna prednost MC metoda je njena procjena vrijednosti za svako stanje koja je neovisna od ostalih - kao što je slučaj kod DP-a. Dalje, računalna cijena procjene jednog stanja **ne** ovisi o ukupnom broju stanja MDP-a - zbog ovoga su MC metode poželjne kada je bitna samo vrijednost jednog ili manjeg podskupa stanja.

3.2 Učenje s vremenskom razlikom

Ideja koja se pokazala kao centralna za PU je učenje s vremenskom razlikom, eng. temporal-difference (TD) learning. TD je kombinacija ideja MC učenja i DP-a. Kao i MC, TD metode mogu učiti direktno iz iskustva bez modela koji opisuje dinamiku okoline. Poput DP-a, TD metode ažuriraju procjene dijelom na temelju unaprijed određenih procjena, ne čekajući na konačan rezultat. Odnos između TD-a, DP-a i MC-s je stalna tema diskusije u teoriji PU-a.

I TD i MC metode koriste iskustvo da bi riješili problem predviđanja. Ako je dostupno iskustvo dobiveno praćenjem strategije π, obje metode ažuriraju svoje pretpostavke o procjeni V vrijednosti v_π za neterminalno stanje S_t koje se pojavilo u tom iskustvu. MC metode čekaju do kraja kako bi postigle *dobit* koju zatim upotrijebe za poboljšanje $V(S_t)$. MC metodu za nestacionarne probleme opisuje se pravilom ažuriranja [2]

$$V(S_t) \leftarrow V(S_t) + \alpha[G_t - V(S_t)]$$

gdje je konstanta α tzv. *parametar veličine koraka*. Dok MC metode čekaju do kraja epizode da bi odredile inkrement za $V(S_t)$ (znači tek kada je poznat G_t), TD metode čekaju samo do idućeg koraka. U trenutku $t+1$ odmah uz pomoć dobivene nagrade R_{t+1} i procjene za $V(S_{t+1})$ rade pretpostavku. Najjednostavniji oblik TD-a koristi pravilo ažuriranja

$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$$

čim prijeđe u stanje S_{t+1} i primi nagradu R_{t+1}. Znači, dok MC čeka na ukupnu dobit G_t, TD koristi procjenu $R_{t+1} + \gamma V(S_{t+1})$. Ova TD metoda poznata je kao TD *jednog koraka* ili TD(0).

[2] Primijenimo inkrementalni izračun aritmetičke sredine iz fusnote (1)

Algoritam 5: *TD(0) za procjenu vrijednosti* $V \approx v_\pi$

Dana je strategija π koju treba procijeniti

Inicijaliziraj: za sve $s \in \mathcal{S}$ (nasumično) postavi $V(s) \in \mathbf{R}$;

Odaberi parametar algoritma $\alpha \in \langle 0, 1math]$;

dok *true (za svaku epizodu)* **čini**

- Inicijaliziraj S:
- **za** *svaki korak epizode:* **čini**
  - $A \leftarrow$ akcija dana od π za stanje S;
  - poduzmi akciju A i promatraj R, S';
  - $V(S) \leftarrow V(S) + \alpha[R + \gamma V(S') - V(S)]$;
  - $S \leftarrow S'$;
  - dok ne dođe do terminalnog S;
- **kraj**

kraj

Korisno je uočiti izraz unutar zagrada u pravilu ažuriranja TD(0) koji je moguće protumačiti kao neku vrstu pogreške: mjeri se razliku između procijenjene vrijednosti za S_t i bolje procjene $R_{t+1} + V(S_{t+1})$. Taj izraz, tzv. *pogreška TD-a*

$$\delta_t = R_{t+1} + V(S_{t+1}) - V(S_t)$$

se u raznim oblicima pojavljuje u svim područjima PU-a.

3.2.1 Primjer: vožnja kući

Svatko tko svakodnevno vozi kući s posla pokušava predvidjeti koliko je vremena potrebno za prelaženje tog puta. U trenutku odlaska s radnog mjesta, zabilježe se vremenske prilike, dan u tjednu, prognoza vremena i sve što bi moglo biti važno. Na primjer, Petar je prošli petak krenuo točno u 6 popodne i pretpostavlja da će mu trebati 30 minuta do kuće. Sjeda u auto u 6:05 i primjećuje da počinje padati kiša. Budući da je vožnja u kišnim uvjetima otežana, Petar predviđa da će mu od tog trenutka do kuće trebati 35 minuta, tj. da će cijeli put trajati ukupno 40 minuta. Međutim, 15 minuta kasnije dok je izlazio s brze ceste pokaže se da je taj dio puta prošao prilično glatko pa smanjuje predviđeno ukupno trajanje puta na 35 minuta. Nažalost, ispred njega vozi spori kamion na jednotračnoj cesti sa zabranom pretjecanja i Petar mora voziti iza njega sve dok ne skrene u svoju ulicu u 6:40. Za tri minute stiže kući. Niz stanja, vremena i pretpostavki vide se u tablici [2].

Stanje	*Proteklo vrijeme (u minutama)*	*Predviđeno vr. preostalo*	*Predviđeno vr. ukupno*
Petar kreće iz ureda, petak u 6	0	30	30
Petar dolazi do auta, pada kiša	5	35	40
Petar silazi s brze ceste	20	15	35
jedna traka, iza kamiona	30	10	40
Petar ulazi u svoju ulicu	40	3	43
Petar dolazi kući	43	0	43

Nagrade u ovome primjeru su proteklo vrijeme za svaki dio puta[3] Budući da je izostavljena korekcija ($\gamma = 1$) dobit svakog stanja je stvarno vrijeme koje je preostalo iz njega. Vrijednost svakog stanja je predviđeno preostalo vrijeme. U trećem stupcu su trenutne procjene za svako od stanja. Najjednostavniji način prikaza MC metode je graf predviđenog ukupnog vremena (zadnji stupac) na nizu stanja, kao sto je prikazano na Slici 3.2.1 (lijevo) - crvene strelice pokazuju promjenu u predviđanju MC metode, one naime točno odgovaraju pogrešci između procijenjene vrijednosti (procijenjenog preostalog vremena) u svakom stanju i stvarne dobiti (stvarnog preostalog vremena). Na primjer, kod izlaska s brze ceste Petar je mislio da će mu trebati još 15 minuta, a zapravo mu je trebalo 23. Ako se primijeni pravilo $V(S_t) \leftarrow V(S_t) + \alpha[G_t - V(S_t)]$ na ovaj trenutak povećat će se vrijednost ovoga stanja, tj predviđenog preostalog vremena. Stvarna greška $G_t - V(S_t)$ u ovome stanju iznosi 8 minuta. Pod pretpostavkom da se koristi parametar veličine koraka $\alpha = 0.5$, onda se predviđeno vrijeme u tom stanju ažurira povećanjem za 4 minute kao posljedica ovoga iskustva. Ovo je možda pre-

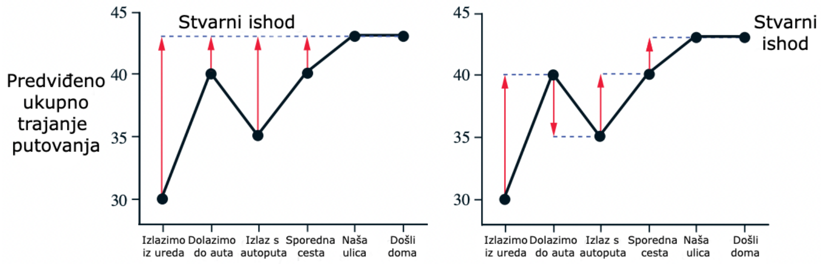
[3] Da se radi o optimizaciji cilj bi bio minimizirati vrijeme putovanja i u tom slučaju bi nagrade bile negativne za količinu proteklog vremena. S obzirom na to da u ovom primjeru fokus nije na optimizaciji, radi jednostavnosti prikaza koriste se pozitivni brojevi.

velika promjena jer se najvjerojatnije neće često ponavljati stanje sa sporim kamionom. U svakom se slučaju ažuriranje vrši tek na kraju epizode, tj. pri dolasku kući - tek je tada poznata stvarna dobit od svakog stanja. Je li nužno čekati kraj epizode da bi se postigao trenutak učenja iz novog stanja? Recimo da neki drugi dan Petar napušta ured i predviđa da će mu trebati 30 minuta, međutim taj dan zapne u ogromnoj gužvi i 25 minuta nakon napuštanja ureda još uvijek se sporo kreće u koloni. U tom trenutku Petar predviđa da mu preostaje još bar 25 minuta putovanja, znači ukupno 50. Dok čeka u koloni već zna da je njegovo početno predviđanje od 30 minuta bilo preoptimistično. Po MC treba čekati do kraja jer stvarna dobit još uvijek nije poznata.

Po TD-u se odmah uči i mijenja početno predviđanje sa 30 na 50 minuta. Zapravo se svako predviđanje pomiče. U prethodnom primjeru vožnje od petka, na slici 3.2.1 (desno) se vide promjene predviđanja koje daje TD pravilo $V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$ (za $\alpha = 1$). Osim što se na ovaj način Petar ima čime zabavljati dok čeka u koloni, postoji još nekoliko računalnih prednosti učenja baziranog na trenutnim pretpostavkama umjesto čekanja do terminalnog stanja da bi se saznala stvarna dobit - neke od tih prednosti su navedene u daljnjem tekstu.

3.3 MC vs. TD

Očita prednost TD metoda nad MC metodama je da su implementirane na inkrementalan način. Kod MC metoda potrebno je dočekati kraj epizode, jer se tek onda saznaje kolika je dobit, dok se kod TD-a čeka samo jedan korak. U mnogo slučajeva se ovo pokazalo kao odlučujući faktor jer u brojnim primjenama epizode traju jako dugo, a kod kontinuiranih slučaj uopće nema



Slika 3.2: Preporučene promjene predviđanja u primjeru "vožnja kući" od MC (lijevo) i TD (desno) metoda [1]

epizoda. Uz to neke MC metode moraju ignorirati ili značajno korigirati epizode u kojima su korištene eksperimentalne akcije što značajno usporava učenje - TD metode nisu podložne ovim problemima jer one uče prilikom svakog pomaka bez obzira koje su akcije poduzete kasnije.

Jako je zgodno učiti o jednoj pretpostavci temeljem druge pretpostavke, bez čekanja na stvarni rezultat, postavlja se pitanje može li se u tom slučaju garantirati konvergenicju ka točnom odgovoru. Iako neće biti ovdje dokazano, na svu sreću odgovor je da! Za bilo koju fiksnu strategiju π TD(0) će konvergirati ka v_π ako je parametar veličine koraka dovoljno malen.

Ako i TD i MC metode asimptotski konvergiraju ka ispravnim predviđanjima, sljedeće logično pitanje je koja od njih dođe prije do rezultata, tj. koja metoda brže uči.

Ovo je još uvijek otvoreno pitanje, u smislu da ne postoji formalni dokaz da jedna metoda konvergira brže od druge. Ali u praksi se pokazalo da TD metode u pravilu konvergiraju brže od MC metoda na stohastičkim zadatcima.

U slučaju rada s konačnom količinom iskustva, recimo 10 epizoda ili 100 vremenskih koraka, ustaljena je praksa iterirati po tom iskustvu dok se ne konvergira do odgovora - pokazalo se da metode TD(0) i MC obe konvergiraju, ali prema različitim odgovorima!

3.3.1 Primjer: Petar predviđa

Petar koji je došao kući u ovom je primjeru prediktor koji promatra rezultate za dane akcije:

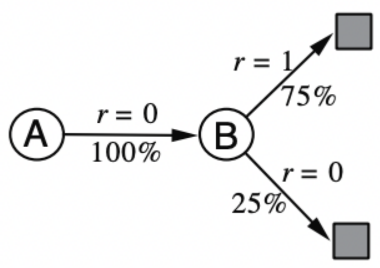
A, 0, B, 0	B, 1
B, 1	B, 1
B, 1	B, 1
B, 1	B, 0

Prva epizoda počinje u stanju A, uz nagradu 0 prelazi u stanje B i onda uz nagradu 0 prelazi u terminalno stanje. Idućih šest epizoda direktno iz stanja B prelaze u terminalno stanje uz nagradu 1 i zadnja epizoda terminira uz nagradu 0. Iz danog skupa podataka koje su pretpostavljene vrijednosti stanja tj. koliki su $V(A)$ i $V(B)$?

Nema nikakvih značajnih dilema oko pretpostavke $V(B) = \frac{6}{8} = \frac{3}{4}$. Međutim, što reći o $V(A)$ za dani skup podataka?

Postoje 2 razumna odgovora. Jedan je kada Petar primijeti da u 100% slučajeva iz stanja A prelazi u stanje B (uz nagradu 0), a budući da je $V(B) = \frac{3}{4}$, to je onda i $V(A) = \frac{3}{4}$. Ovaj se odgovor može promatrati kao da se prvo modelira MDP, grafika (3.3.1), i onda se računaju procjene za taj model. Ovo je i odgovor koji bi dao TD(0).

Drugi razuman odgovor je da svako promatrano pojavljivanje A završava s dobiti 0, dakle $V(A) = 0$. Ovo je odgovor koji bi dala MC metoda - i odgovor koji daje minimum srednje kvadratne greške na skupu za treniranje. Dapače, greška u ovom slučaju iznosi 0 ali ipak se prvi odgovor čini smisleniji. Ako je proces koji Petar promatra Markovljev, onda će prvi odgovor dati



Slika 3.3: Jednostavan model MDP-a [1]

manju grešku na budućim podatcima - iako je MC odgovor bolji za postojeće podatke.

Poglavlje 4

Kontrola bez modela

Sve prethodno napisano bilo je svojevrstan uvod u ovo poglavlje koje je zapravo centralni dio ovog rada. Konačno će se moći izvući zaključci o načinu na koji neki robot, bačen u nepoznato okruženje, otkriva što treba napraviti - kako maksimizirati svoju nagradu, tj. birati akcije koje će dugoročno donijeti najveću dobit bez da unaprijed zna išta o okolišu u kojem se nalazi. Sve druge tehnike PU, koje zbog sadržajnog ograničenja nisu spomenute u ovom radu, temelje se na ovome.

U prethodnom je poglavlju predstavljen set potrebnih alata- predviđanje bez modela, tj. kako vrednovati danu strategiju. Pomoću toga je moguće procijeniti vrijednosnu funkciju strategije u nepoznatom MDP-u. Taj osnovni alat će koristiti za metode kontrole, tj. unapređenju kako bi došli do optimalne vrijednosne funkcije i optimalne strategije.

Većina zanimljivih problema PU-a kao npr. botovi koji igraju računalne igrice, savijanje proteina, upravljanje fuzijskom reakcijom itd. imaju MDP, tj. mogu se opisati kao MDP-ovi - imaju stanja okoline i neka pravila/logiku po kojoj se ponašaju. Međutim, ili je taj MDP nepoznat, nepoznata su pravila ponašanja okoline pa se oslanja na metode učenja bez modela, ili je

MDP poznat ali je toliko složen da ga je nepraktično koristiti za bilo što osim uzorkovanja - u tom slučaju se opet radi o području učenja bez modela kod kojeg se uči iz iskustva pokušavajući razne stvari. Dobar primjer potonjeg je stvarni svijet - i kad bi znali točna pravila stvarnog svijeta (vjerojatno neka teorija kaosa), neizmjerno je jednostavnije uzeti uzorak - napraviti eksperiment - i vidjeti što se dogodilo nego računati ponašanje svakog atoma.

4.1 Monte-Carlo kontrola

Ako se na tren opet obrati pažnja na generaliziranu iteraciju strategija (GPI), njezina je osnovna ideja alterniranje dva procesa: *vrednovanje strategije* i poboljšanje strategije. Ovaj proces konvergira ka optimalnoj vrijednosnoj funkciji i optimalnoj strategiji. Postavlja se pitanje što je potrebno poduzeti kako bi se upotrebom ta dva procesa dobila kontrola bez modela.

Najjednostavnije bi bilo iskoristiti MC predviđanje za vrednovanje strategije i pohlepno poboljšanje strategije - međutim pojavljuje se nekoliko poteškoća. MC predviđanje uzme određen broj uzoraka i računa aritmetičke sredine dobiti kako bi aproksimirali vrijednosnu funkciju strategije, zatim se generira nova strategija koja se ponaša pohlepno u odnosu na procijenjenu vrijednosnu funkciju. U ovom su se slučaju pojavila dva problema:

Bez modela: kako je moguće biti bez modela ako se koristi funkciju v_π. Ako je poznata samo vrijednosna funkcija stanja i koristi se pohlepni algoritam s obzirom na nju, iz (2.3) tj.

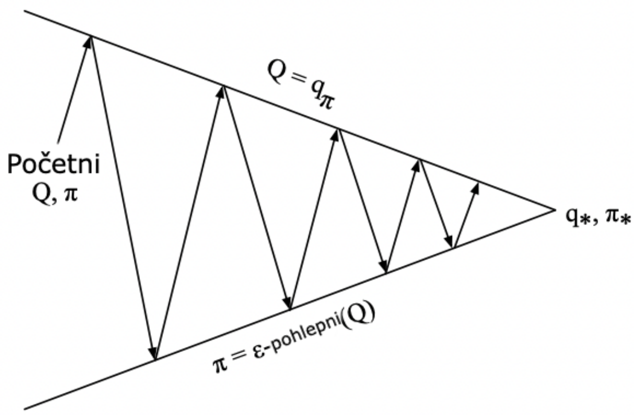
$$\pi'(s) = \underset{a}{\operatorname{argmax}} \sum_{s',r} \mathcal{P}_0(s', r|s, a)[r + \gamma v_\pi(s')]$$

vidi se da je i dalje potreban model okoline - tj. matrica prijelaza $\mathcal{P}_0$, koje nema! Alternativa je korištenje vrijednosne funkcije akcije q_π jer u tom slučaju je dovoljno tražiti maksimum po Q vrijednostima

$$\pi'(s) = \underset{a \in \mathcal{A}}{\operatorname{argmax}}\, Q(s, a)$$

Ovo omogućuje kontrolu u slučaju da nema informacije o modelu - ako se zna informacija q i želi se raditi pohlepno poboljšanje strategije u svakom stanju bira se akciju koja maksimizira q vrijednosti.

Problem istraživanja: ako je dana strategija koja se slijedi pohlepna, onda nema nikakve garancije da putanje kojima se prolazi pokrivaju cijeli prostor stanja - lako se može dogoditi da postoje dijelovi prostora stanja koji imaju veći potencijal, daju bolje nagrade, no do kojih agent nikad neće ni doći. Kratak primjer će poslužiti kao ilustracija: Petar se nalazi ispred dvaju vrata iza kojih se nalaze različite **stohastičke** nagrade. On otvori lijeva vrata i dobije nagradu 0, tj. $V(lijevo) = 0$. Nulom nije baš oduševljen pa isproba drugi put desna vrata i dobije nagradu +1, tj. $V(desno) = +1$. Dalje se ponaša *pohlepno* pa opet otvara desna vrata i dobije nagradu +3 pa je $V(desno) = +2$ itd. On otvara desna vrata i dobija neku pozitivnu nagradu koja varira između 1 i 3. Međutim, je li stvarno odabrao najbolja vrata? - problem je što mu zapravo nije poznata vrijednost lijevih vrata jer ih je probao otvoriti samo jedan put. Dakle, treba nastaviti istraživati sve kako bi bio siguran da zna prave vrijednosti svih opcija koje mu se nude. Ako prestane istraživati određene akcije može kasnije donijeti krive odluke i zapeti u nekom lokalnom optimumu. Kako bi se ovaj problem rješio, uz garanciju da se uvijek istražuje sva stanja i sve akcije, koristi se dosta jednostavna ideja - iako ima mnogo sofisticiranijih metoda jako je teško postići bolje rezultate od nje. Koristi se tzv. $\epsilon -$ *pohlepno istraživanje*. Svaki put kada agent bira akciju sa vjerojatnošću ϵ izabrat će neku nasumičnu akciju, a sa vjerojatnošću $1 - \epsilon$ bira pohlepnu akciju. Dakle, uglavnom se prati pohlepnu strategiju ali s malom vjerojatnošću se vrše i druge akcije. Možda se čini naivno ali ima neka dobra svojstva. Garantira se istraživanje svih akcija i poboljšavanje strategije. Lako se pokaže (vidi [1]) da $\epsilon-$*pohlepno istraživanje* daje poboljšanje strategije i može se koristiti u spomenutoj verziji GPI-a [1].



Slika 4.1: Monte-Carlo iteracija strategija [1]:
Vrednovanje strategije: MC vrednovanje, $Q = q_\pi$
Poboljšanje strategije: $\epsilon - pohlepno$

4.2 Kontrola s vremenskom razlikom

Kod učenja MC metode nadovezala se na nju metoda vremenske razlike, TD za razliku od MC-a može se koristiti bez čekanja na kraj epizode ili za kontinuirane slučajeve koji nikad ne dođu do terminalnog stanja. Isprobavanje

[1]Kao i kod DP-a nije nužno do kraja vrednovati strategiju - nekad već nakon nekoliko koraka ima dovoljno informacija za konstruiranje puno bolje strategije, bez potrebe za dodatnim iteracijama kako bi se vrijednosnu funkciju izračunalo do kraja. U ekstremnom slučaju nakon svake epizode (završene putanje) može se raditi iteraciju poboljšanja strategije.

iste ideje kao kod MC-a se zapravo samo po sebi nameće, radi se o generaliziranoj iteraciji strategije. Iz već prethodno navedenih razloga, potrebno je koristiti vrijednosnu funkciju akcije q_π i $\epsilon - pohlepno$ poboljšanje strategije. Na ovaj se način automatski dobije jedan od najboljih algoritama PU-a - jedino poboljšanje koje će se primijeniti je dodatno povećanje frekvencije iteracija. Kako TD ne treba čekati niti do kraja epizode da bi ažurirali vrijednosnu funkciju, dapače može se ažurirati i nakon samo jednog koraka, a uvijek je poželjno koristiti najnovije dostupne vrijednosti kod biranja akcija, onda se može povećati broj iteracija primjenjujući ih na svakom vremenskom koraku.

Generalno pravilo TD učenja je tzv. **SARSA**

$$Q(S,A) \leftarrow Q(S,A) + \alpha(R + \gamma Q(S',A') - Q(S,A))$$

Dakle ako je tražena vrijednost za uređeni par stanja i akcije (S, A). Iz stanja S agent vrši akciju A i okolina dodjeljuje nagradu R te agent završava u stanju S'. Zatim slijedeći odabranu strategiju agent vrši akciju A' tako da se u dva koraka dobije$(S, A) \xrightarrow{R} (S', A')$ odakle naziv SARSA. Počinje se od odabrane Q vrijednosti i pomiče se u smjeru koji daje nagrada uvećana za razliku korigirane vrijednosti idućeg stanja i vrijednosti trenutnog stanja.

Algoritam 6: *SARSA za procjenu* $Q \approx q_*$

Podatci: parametri algoritma: veličina koraka $\alpha \in \langle 0, 1]$, maleni $\epsilon > 0$

Inicijaliziraj: $Q(s, a)$ za sve $s \in \mathcal{S}$, $a \in \mathcal{A}$ po volji, osim $Q(terminal, \cdot) = 0$

za *svaku epizodu* **čini**

  Inicijaliziraj $\mathcal{S}$;

  Odaberi A u stanju S koristeći strategiju izvedenu iz Q (npr. $\epsilon - pohlepno$);

  dok *S nije terminalan, za svaki korak epizode* **čini**

    Poduzmi akciju A i promotri R i S';

    Odaberi A' u stanju S' koristeći strategiju izvedenu iz Q (npr. $\epsilon - pohlepno$);

    $Q(S, A) \leftarrow Q(S, A) + \alpha(R + \gamma Q(S', A') - Q(S, A))$;

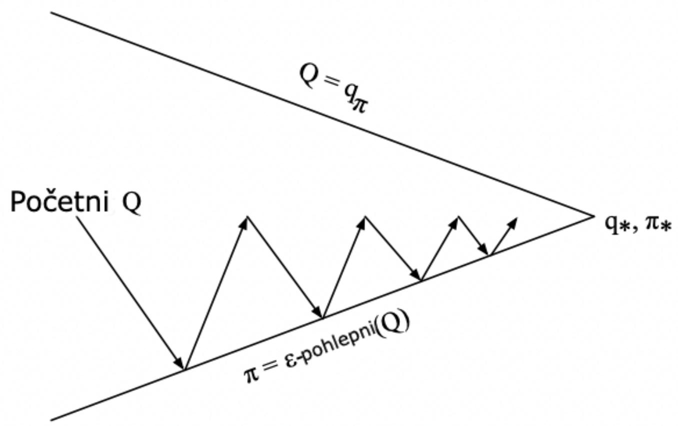
    $S \leftarrow S'$;

    $A \leftarrow A'$;

  kraj

kraj

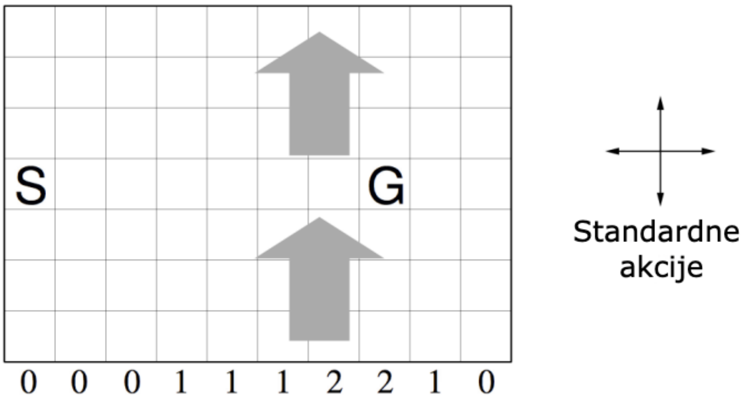
Drugim riječima: U nekom stanju se razmatra o poduzimanju određene akcije. Ako se ta akcija poduzme i promotri se dobivena nagrada i vrijednost iduće akcije koju će agent poduzeti, taj postupak daje procjenu vrijednosti strategije koja se slijedi. Ta se procjena koristi za ažuriranje vrijednosti para stanja i akcije iz kojih se započelo - ova generalna ideja je zapravo već viđena nekoliko puta i trebala bi odmah podsjetiti na Bellmanove jednadžbe.



Slika 4.2: TD iteracija strategija (nakon **svakog** vremenskog koraka) [1]:
Vrednovanje strategije: SARSA Q_π
Poboljšanje strategije: $\epsilon - pohlepno$

4.2.1 Primjer: vjetroviti Mrežni svijet [1]

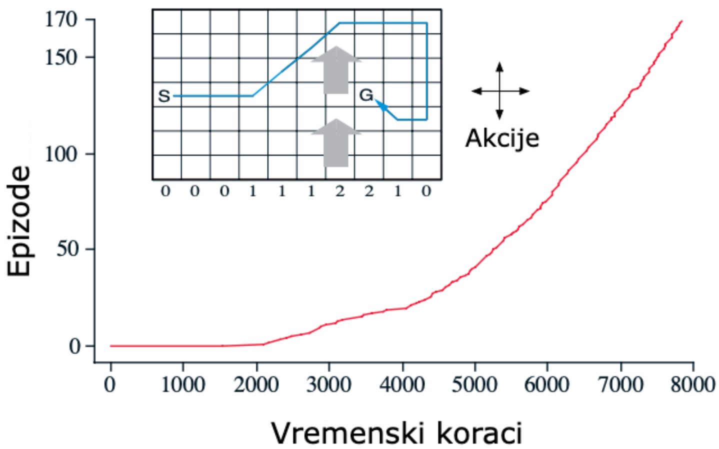
Na slici (4.2.1) je primjer Mrežnog svijeta s naznačenim početnim (S) i konačnim (G) stanjem te jednim dodatkom: postoji "vjetar" koji puše od dolje prema gore sredinom tablice. Akcije agenta su standardne (gore, dolje, lijevo i desno), ali ako prolazi sredinom tablice vjetar, tj. okolina, ga pomiče prema gore što utječe na stanje u kojem će završiti nakon akcije - jačina vjetra ovisi o stupcu a može se vidjeti ispod tablice. Na primjer, ako se agent nalazi u polju desno od cilja i krene prema lijevo, završit će u polju neposredno iznad cilja. Za svaki korak nagrada je -1 sve dok agent ne dođe do cilja. Na grafu (4.2.1) se vidi rezultat primjene $\epsilon - pohlepne$ SARSA-e na ovaj primjer sa $\epsilon = 0.1$ i $\alpha = 0.5$ te početnim vrijednostima $Q(s, a) = 0$ za sve s i a. Iz grafa se iščitava da je za završetak prve epizode bilo potrebno više od 2000 koraka. Međutim, nakon prve epizode druga je puno brža - dok se dođe do 8000 učinjenih vremenskih koraka već je odavno otkrivena



Slika 4.3: Vjetroviti Mrežni svijet

optimalna strategija (vidi se na slici) ali zbog $\epsilon - pohlepnog$ istraživanja je prosječna duljina epizode 17 koraka - za dva više od optimalnih 15.

MC metode ne koriste u ovome slučaju jer ne postoji garancija za strategiju da će doći do kraja epizode, npr. kad bi strategija uputila agenta da stalno stoji na istom mjestu. SARSA nema ovih problema jer uči tijekom same epizode pa mijenja danu strategiju s drugačijom svaki put kad nauči nešto novo.



Slika 4.4: Rješenje vjetrovitog Mrežnog svijeta & Graf potrošenih vremenskih koraka i prijeđenih epizoda [1]

Poglavlje 5

Praktični primjer: Blackjack

Kako bi se nadišlo puko teoretiziranje i navođenje primjera iz knjige, primjenjivost metode je iskušana u "stvarnom svijetu" primjenom par linija koda. Kod pažljivijeg iščitavanja rada, oštro oko je sigurno primijetilo nedostatak pratećeg primjera u poglavlju 4.1 **Monte-Carlo kontrola** - ovo će poglavlje to ispraviti. Kao što i sam naslov poglavlja sugerira, traži se optimalna strategija za igranje Blackjacka. Budući da su pravila i algoritam MC kontrole već poznati, odmah ide modeliranje problema.

Program je u svrhu ovog rada napisan u *vanilla* JavaScript-u i pokretan je uz pomoć Node.js-a v16.13.1 , no očigledno je toliko jednostavan da se lako da implementirati u bilo kojem programskom jeziku i pokrenuti u većini okolina. Program je radi preglednosti podijeljen u dvije datoteke: *pomocni.js* - modul koji sadrži definicije svih funkcija koje se koriste i *program.js* - koji sadrži lako čitljivu implementaciju algoritma. Napokon evo i koda:

```
\\                                  10 J  Q  K
const KARTE = "A,2,3,4,5,6,7,8,9,10,10,10,10".split(',');
const izvuciKartu = () => KARTE[Math.floor(Math.random() * KARTE.length)];
```

```

const inkrementalnoIzracunajProsjek = (stariProsjek, novaVrijednost, n) => stariProsjek + (novaVrijednost - stariProsjek) / (n + 1);

const resetirajNovuTablicuVrijednosti = () => ({
  sAsom: [...Array(10)].map((el, i) =>
    [...Array(10)].map((elInner) => ({
      h: { val: 0, brojPonavljanja: 0 },
      s: { val: 0, brojPonavljanja: 0 }
    }))
  ),
  bezAsa: [...Array(10)].map((el, i) =>
    [...Array(10)].map((elInner) => ({
      h: { val: 0, brojPonavljanja: 0 },
      s: { val: 0, brojPonavljanja: 0 }
    }))
  )
});
```

Listing 5.1: pomocni.js 1/3

Većina koda je samoobjašnjiva, *resetirajNovuTablicuVrijednosti* vraća objekt sa dvije 10x10 tablice u koje će se spremati prosjeci ostvarenih dobiti iz danog stanja za trenutnu strategiju, tj. vrijednosti tih stanja.

```
const zapocniPartiju = () => {
  let iskoristivAS = false;
  let suma = 0;

  while (suma < 12) {
    const izvucenaKarta = izvuciKartu();
    if (izvucenaKarta === "A" && !iskoristivAS) {
      suma += 10;
```

```
      iskoristivAS = true;
    } else if (izvucenaKarta === "A" && iskoristivAS) {
      suma += 1;
    } else {
      suma += +izvucenaKarta;
    }
  }

  return {
    suma,
    iskoristivAS,
    putanja: [],
    dobit: undefined,
    protivnikPokazuje: izvuciKartu()
  };
};

const poduzimamoAkciju = (partija, izvucenaKarta) => {
  partija.putanja.push({
    suma: partija.suma,
    odabranaAkcija: "h",
    iskoristivAS: partija.iskoristivAS
  });
  if (izvucenaKarta === "A") {
    partija.suma += 1;
  } else {
    partija.suma += +izvucenaKarta;
  }

  if (partija.suma > 21 && partija.iskoristivAS) {
    partija.suma -= 9;
    partija.iskoristivAS = false;
  }
```

```
};

const odigrajEpsilonPohlepno = (partija, epsilon,
    staraTablicaVrijednosti) => {
  let odabranaAkcija = undefined;

  while (odabranaAkcija !== "s" && partija.suma < 22) {
    const trenutnoStanje = (partija.iskoristivAS
      ? staraTablicaVrijednosti.sAsom
      : staraTablicaVrijednosti.bezAsa)[partija.suma - 12][
      partija.protivnikPokazuje === "A" ? 0 : +partija.
          protivnikPokazuje - 1
    ];
    const [preferiranaAkcija, nePreferiranaAkcija] =
      trenutnoStanje === 'h' ? ["h", "s"] : ["s", "h"];

    odabranaAkcija =
      (Math.random() > epsilon && preferiranaAkcija) ||
          nePreferiranaAkcija;

    if (odabranaAkcija === "h") {
      poduzimamoAkciju(partija, izvuciKartu());
    } else {
      partija.putanja.push({
        suma: partija.suma,
        odabranaAkcija,
        iskoristivAS: partija.iskoristivAS
      });
    }
  }
};
```

Listing 5.2: pomocni.js 2/3

Izvlače se karate sve dok nije postignuta suma od barem 12 jer ako je suma 11 ili manje nema se što izgubiti, tj. nema nikakve dileme da je optimalna strategija uzeti još jednu kartu. Kuća ima samo jednu kartu koju igrač vidi, jer karta koju ne vidi nema nikakav utjecaj na njegove odluke pa je po potrebi izvlači tek nakon što igrač završi.

```
const kucaIgra = (partija) => {
  if (partija.suma > 21) {
    partija.dobit = -1;
    return;
  }
  let kucaImaIskoristivogAsa = partija.protivnikPokazuje === "A";
  let sumaKuce =
    partija.protivnikPokazuje === "A" ? 10 : +partija.protivnikPokazuje;

  while (sumaKuce < 17) {
    const izvucenaKarta = izvuciKartu();
    if (izvucenaKarta === "A") {
      sumaKuce += kucaImaIskoristivogAsa ? 1 : 10;
      kucaImaIskoristivogAsa = kucaImaIskoristivogAsa || true;
    } else {
      sumaKuce += +izvucenaKarta;
    }

    if (sumaKuce > 21 && kucaImaIskoristivogAsa) {
      sumaKuce -= 9;
      kucaImaIskoristivogAsa=false;
    }
  }
```

```
  if (sumaKuce > 21) {
    partija.dobit = 1;
    return;
  }

  if (sumaKuce === partija.suma) {
    partija.dobit = 0;
    return;
  }
  if (partija.suma > sumaKuce) {
    partija.dobit = 1;
    return;
  } else {
    partija.dobit = -1;
    return;
  }
};

const azurirajStanjaNoveTablice = (novaTablicaVrijednosti,
    partija) => {
  partija.putanja.forEach((stanje) => {
    const stanjeUTablici = (stanje.iskoristivAS
      ? novaTablicaVrijednosti.sAsom
      : novaTablicaVrijednosti.bezAsa)[stanje.suma - 12][
      partija.protivnikPokazuje === "A" ? 0 : +partija.
          protivnikPokazuje - 1
    ][stanje.odabranaAkcija];
    stanjeUTablici.val = inkrementalnoIzracunajProsjek(
      stanjeUTablici.val,
      partija.dobit,
      stanjeUTablici.brojPonavljanja
    );
    stanjeUTablici.brojPonavljanja += 1;
```

```
  });
};

const formirajNovuTablicu = (novaTablica) => {
  const formirana = {};
  formirana.sAsom = novaTablica.sAsom.map((redak) =>
    redak.map(({ h, s }) => h.val >= s.val ? 'h': 's' )
  );

  formirana.bezAsa = novaTablica.bezAsa.map((redak) =>
    redak.map(({ h, s }) => h.val >= s.val ? 'h': 's')
  );
  return formirana;
};

const ispirintajTablicu = (tablica) => {
  console.log('s Asom:');
  console.log('     A   2   3   4   5   6   7   8   9  10')
  tablica.sAsom.forEach((redak, i) => console.log(i+12, JSON.stringify(redak)))
  console.log('================================');
  console.log('bez Asa:')
  console.log('     A   2   3   4   5   6   7   8   9  10')
  tablica.bezAsa.forEach((redak, i) => console.log(i+12, JSON.stringify(redak)))
}

module.exports = {
  resetirajNovuTablicuVrijednosti ,
  ispirintajTablicu,
  azurirajStanjaNoveTablice,
  zapocniPartiju,
  poduzimamoAkciju,
```

```
  odigrajEpsilonPohlepno,
  kucaIgra,
  formirajNovuTablicu
}
```

Listing 5.3: pomocni.js 3/3

- Funkcija *kucaIgra* odradi izvlačenje karata za kuću, nastavlja izvlačiti dok ne postigne barem 17 i određuje dobiti koje su poznate čim kuća završi.
- *azuriranjeStanjaNoveTablice* računa i postavlja nove procjene vrijednosti posjećenih stanja u danoj partiji (kako broj posjeta tih stanja raste, procjena će težiti stvarnoj vrijednosti tih stanja).
- *formirajNovuTablicu* iz tablice vrijednosti formira tablicu koja govori kako se ponaša pohlepna strategija s obzirom na te vrijednosti.

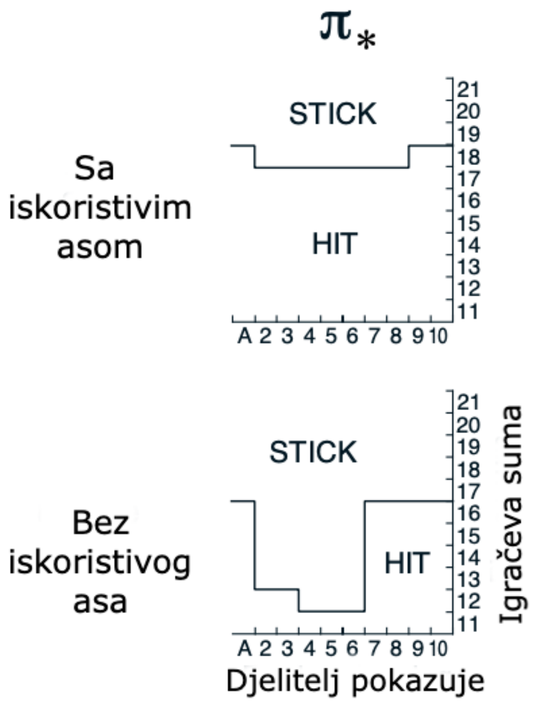
```
const {
  resetirajNovuTablicuVrijednosti,
  ispirintajTablicu,
  formirajNovuTablicu,
  zapocniPartiju,
  odigrajEpsilonPohlepno,
  kucaIgra,
  azurirajStanjaNoveTablice
} = require('./pomocni')

const EPSILON = 0.1;

let staraTablicaVrijednosti = {
  sAsom: [...Array(10)].map((_, i) =>[...Array(10)].map(() =>
      i + 12 < 17 ? 'h': 's')),
```

```
  bezAsa: [...Array(10)].map((_, i) =>[...Array(10)].map(() => i + 12 < 17 ? 'h': 's'))
};
let novaTablicaVrijednosti = resetirajNovuTablicuVrijednosti();
let partija;
let brojac = 0;

do {

  if(brojac !== 0) staraTablicaVrijednosti = formirajNovuTablicu(novaTablicaVrijednosti);
  novaTablicaVrijednosti = resetirajNovuTablicuVrijednosti();

  for (let i = 0; i < 10000000; i++) {
    partija = zapocniPartiju();
    odigrajEpsilonPohlepno(partija, EPSILON, staraTablicaVrijednosti);
    kucaIgra(partija);
    azurirajStanjaNoveTablice(novaTablicaVrijednosti, partija);
  }

  console.log('brojac:', brojac)
  brojac++
} while(JSON.stringify(staraTablicaVrijednosti) !== JSON.stringify(formirajNovuTablicu(novaTablicaVrijednosti)))

ispirintajTablicu(formirajNovuTablicu(novaTablicaVrijednosti))
```

Listing 5.4: program.js

Dakle, import-aju se sve funkcije koje će se koristiti iz *pomocni.js* i postavlja se vrijednost epsilona (u ovom konkrentno slučaju 0.1, tj. u prosjeku svaku desetu akciju se iskoristi za istraživanje novih stanja). Zatim se definira *staruTablicuVrijednosti* kojom se utvrđuje početnu strategiju (u ovom je primjeru za početnu strategiju odabrana ista strategija koju koristi kuća, a iz teorije je poznato da od koje god strategije počeli uvijek će se na kraju doći do optimalne). U *do-while* petlji može se vidjeti implementaciju GPI-a gdje *for* petlja služi za vrednovanje strategije.

Rezultati strategije su sljedeći:



Slika 5.1: Rezultat kojeg su dobili Sutton i Barto [2]

Kod usporedbe primjera iz knjige i rezultata ovog jednostavnog programa predstavljenog u ovom radu, rezultati su prilično zadovoljavajući budući da su ostvarili prvotno zamišljenu svrhu. Kod stanja s iskoristivim asom dobije se identičnu strategiju, a kod stanja bez iskoristivog asa se razlikuju u 3 (od 100) stanja - ovo je posljedica činjenice da se partije stvarno odvijaju nasumično. 10 milijuna odigranih partija je velik broj, ali i dalje je značajno manji od beskonačno.

Program je kod testiranja pokrenut oko 100 puta i u pravilu se rezultat razlikovao u jednom ili dva stanja od ovoga prikazanog na slici, a nerijetko je rezultat bila i optimalna strategija koju su postigli Sutton i Barto. Ono što se značajno razlikuje od izvođenja do izvođenja je broj potrebnih iteracija: od nekoliko iteracija do 100+. Ali kad se oslabi uvjet na *do-while* petlji tako da stane kada se strategije razlikuju npr. manje ili jednako 2 stanja, program u pravilu stane u manje od 5 iteracija - pomoćne funkcije za implementaciju ovoga pristupa mogu se vidjeti na sljedećem GitHub profilu `https://github.com/Ozra/Diplomski`. Ovim je programom razriješeno pitanje nalaženja optimalne strategije u nepoznatoj okolini pomoću e-pohlepnog istraživanja, poboljšanja strategije i metodama vremenske razlike.

```
petarozretic@MacBook-Pro-2 Diplomski % node program.js
brojac: 0
brojac: 1
brojac: 2
brojac: 3
brojac: 4
brojac: 5
brojac: 6
brojac: 7
brojac: 8
brojac: 9
brojac: 10
brojac: 11
brojac: 12
brojac: 13
brojac: 14
s Asom:
     A   2   3   4   5   6   7   8   9  10
21 ["s","s","s","s","s","s","s","s","s","s"]
20 ["s","s","s","s","s","s","s","s","s","s"]
19 ["s","s","s","s","s","s","s","s","s","s"]
18 ["h","s","s","s","s","s","s","s","h","h"]
17 ["h","h","h","h","h","h","h","h","h","h"]
16 ["h","h","h","h","h","h","h","h","h","h"]
15 ["h","h","h","h","h","h","h","h","h","h"]
14 ["h","h","h","h","h","h","h","h","h","h"]
13 ["h","h","h","h","h","h","h","h","h","h"]
12 ["h","h","h","h","h","h","h","h","h","h"]
================================
bez Asa:
     A   2   3   4   5   6   7   8   9  10
21 ["s","s","s","s","s","s","s","s","s","s"]
20 ["s","s","s","s","s","s","s","s","s","s"]
19 ["s","s","s","s","s","s","s","s","s","s"]
18 ["s","s","s","s","s","s","s","s","s","s"]
17 ["s","s","s","s","s","s","s","s","s","s"]
16 ["h","s","s","s","s","s","h","h","h","s"]
15 ["h","s","s","s","s","s","h","h","h","h"]
14 ["h","s","s","s","s","s","h","h","h","h"]
13 ["h","s","s","s","s","s","h","h","h","h"]
12 ["h","s","s","s","s","s","h","h","h","h"]
petarozretic@MacBook-Pro-2 Diplomski %
```

Slika 5.2: Rezultat izvođenja programa: nakon 15 iteracija rezultat je optimalna strategija. U redu iznad matrica se nalazi karta koju pokazuje kuća, a s lijeve strane suma koju igrač ima.

Poglavlje 6

Zaključak

Podržano učenje zadnjih godina puni naslovnice medija: od pobjeđivanja prvaka u Go ili profesionalaca u Starcraftu II do slaganja proteina i kontroliranja plazme u fuzijskom rekatoru. U ovome radu, zbog sadržajnih ograničenja, obrađene su samo osnove PU-a, ali u dovoljnoj mjeri da bi se mogao naslutiti daljnji razvoj, iako nije izrijekom spomenut. Naime PU u kombinaciji s dubokim neuronskim mrežama, tzv. *Duboko podržano učenje*, tek otključava svoj pravi potecijal. Ovaj je rad zbog svojih unaprijed određenih okvira, postavio skroman temelj koji otvara mogućnosti daljnjem razvoju i nalaženju novih načina primjene podržanog učenja. Samo stotinjak linija koda postiglo je isti rezultat kao E. Thorp koji je svojom knjigom izmijenio kockarsku industriju. Dakle, prostora za rast i napredovanje svakako ima. No, ovaj je rad postigao prvotnu namjeru, doskočiti klasičnim poteškoćama i preprekama koje se pojavljuju kod korištenja različitih metoda PU. Njihovom kombinacijom napisani je program prilično uspješno došao do optimalne vrijednosne funkcije i optimalne strategije za igru Blackjacka. Kroz taj praktični primjer, osim navedenih rješivih prepreka, moglo se naslutiti i otvoreno pitanje odnosa iskorištavanja znanja i istraživanja - možda bi povečanjem epsilona sa 0.1 na

0.2 smanjili prosječan broj iteracija: je li 10 000 000 partija bilo pretjerivanje za vrednovanje strategije; Sutton i Barto su u svom primjeru bili zadovoljni sa 500 000. I u konačnici, ako se želi poboljšati rezultate koji bi se dobili isključivo podržanim učenjem, je li odgovor u kombinaciji podržanog učenja i dubokih neuronskih mreža, odnosno u dubokom PU-u.

Literatura

[1] Silver, D., *Introduction to reinforcement learning*
https://www.youtube.com/playlist?list=PLqYmG7hTraZDM-OYHWgPebj2MfCFzFObQ
https://www.davidsilver.uk/teaching/

[2] Sutton, R.S., Barto, A.G., *Reinforcement learning*
http://incompleteideas.net/book/RLbook2020.pdf

[3] Szepesvári, C., *Algorithms for Reinforcement Learning*
https://sites.ualberta.ca/~szepesva/papers/RLAlgsInMDPs.pdf

[4] Blažević, L., *Podržano učenje i Q-učenje*
https://www.mathos.unios.hr/~mdjumic/uploads/diplomski/BLA42.pdf

[5] Čupić, M., *Umjetna inteligencija: Podržano učenje*
http://java.zemris.fer.hr/nastava/ui/rl/rl-20200401.pdf

[6] Kaparthy, A., *Deep Reinforcement Learning: Pong from Pixels*
https://karpathy.github.io/2016/05/31/rl/

[7] *AlphaGo - The Movie — Full award-winning documentary*
https://www.youtube.com/watch?v=WXuK6gekU1Y

[8] *DeepMind StarCraft II Demonstration*
`https://www.youtube.com/watch?v=cUTMhmVh1qs`

[9] *Accelerating fusion science through learned plasma control*
`https://deepmind.com/blog/article/Accelerating-fusion-science-through-learned-plasma-control`

[10] *AlphaFold: The making of a scientific breakthrough*
`https://www.youtube.com/watch?v=gg7WjuFs8F4`

[11] *DeepMind x UCL — Deep Learning Lecture Series 2021*
`https://www.youtube.com/playlist?list=PLqYmG7hTraZDVH599EItlEWsUOsJbAodm`

TEMELJNA DOKUMENTACIJSKA KARTICA

PRIRODOSLOVNO–MATEMATIČKI FAKULTET
SVEUČILIŠTA U SPLITU
ODJEL ZA MATEMATIKU

DIPLOMSKI RAD

OSNOVE PODRŽANOG UČENJA

Petar Ozretić

Sažetak:

Ovaj se rad bavi osnovama podržanog učenja iznoseći sve poznate metode korištene u PU poput Bellmanovih jednadžbi, Dinamičkog programiranja, teorema poboljšanja strategije,Monte-Carlo učenja, učenja s vremenskom razlikom itd. Osnovni problem je bio nalaženje optimalne strategije uz pomoć optimalne vrijednosne funkcije, i to u okolini bez modela. Kombinacijom navedenih metoda, posebno MC iteracije strategija , napisan je kod koji je uspješno nalazio optimalne strategije na primjeru igre Blackjack.

Ključne riječi:

agent, akcija, strategija, vrijednosna funkcija, model okoline, Markovljevo svojstvo, Bellmanove jednadžbe, Monte Carlo kontrola, učenje s vremenskom razlikom, dinamičko programiranje, pohlepno istraživanje, SARSA vrednovanje strategije

Podatci o radu:

62 stranice, 15 slika i 1 tablica, 11 literaturnih navoda, pisano na hrvatskom jeziku)

Mentor: *izv.prof. dr. sc. Saša Mladenović*

Članovi povjerenstva:

izv.prof. dr. sc. Saša Mladenović

TEMELJNA DOKUMENTACIJSKA KARTICA

doc.dr.sc. Vesna Gotovac Đogaš

dr.sc. Ana Perišić

Povjerenstvo za diplomski rad je prihvatilo ovaj rad *UPISATI DATUM ODOBRENJA OBRANE*

TEMELJNA DOKUMENTACIJSKA KARTICA

FACULTY OF SCIENCE, UNIVERSITY OF SPLIT

DEPARTMENT OF MATHEMATICS

MASTER'S THESIS

BASICS OF REINFORCEMENT LEARNING

Petar Ozretic

Abstract:

This paper deals with the basics of reinforcement learning, presenting all known methods used in PU such as Bellman equations, Dynamic programming, strategy improvement theorem, MC learning, temporal difference learning, etc. The basic problem was finding an optimal strategy that converges to an optimal value function, and in a model-free environment. By combining the above methods, especially MC strategy iteration, a code was written that successfully found optimal strategies on the example of the Blackjack game.

Key words:

agent, action, strategy, value function, environment model, Markov property, Bellman equations, Monte Carlo control, temporal difference learning, dynamic programming, greedy strategy, SARSA strategy

Specifications:

62 pages, 15 figures and 1 table, 11 references, written in Croatian)

Mentor: *Saša Mladenović, PhD, Associate Professor*

Committee:

Saša Mladenović, PhD, Associate Professor

Vesna Gotovac Đogaš, PhD, Assistant professor

Ana Perišić, PhD, Postdoctoral Researcher

TEMELJNA DOKUMENTACIJSKA KARTICA

This thesis was approved by a Thesis commettee on *UPISATI DATUM ODOBRENJA OBRANE*