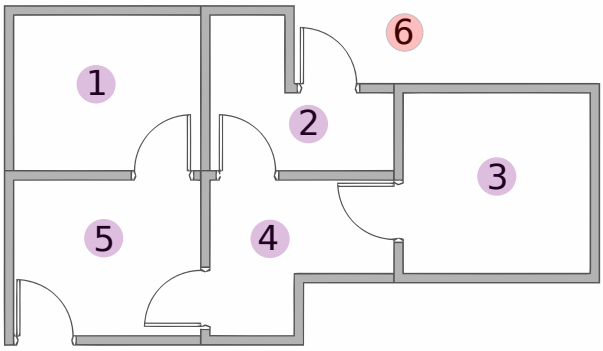
Q Learning

Ivan Androš, Dejan Peretin, Petra Podolski, *PMF - Matematički odsjek, Zagreb*

Svibanj 2011.

Q Learning je jedna od tehnika učenja s podrškom. Agent uči iz svog iskustva bez učitelja, tako da ispituje stanje po stanje sve dok ne dođe do cilja. Svaki takav ciklus istraživanja jednog stanja naziva se epizoda. Konkretno, agent uči evaluacijsku funkciju $Q : S \times A \to \mathbb{R}$ gdje je S skup stanja, a A skup akcija. Poduzimajući akcije, agent se kreće iz stanja u stanje, a za svako stanje "dobiva" nagradu (realni broj) koji pokazuje korisnost stanja. Cilj agenta je maksimizirati nagradu. Konvergencija ovog algoritma se može dokazati. Agentu ne mora biti poznat model okoliša, jer bi se u tom slučaju mogao iskoristiti neki od bržih algoritama.

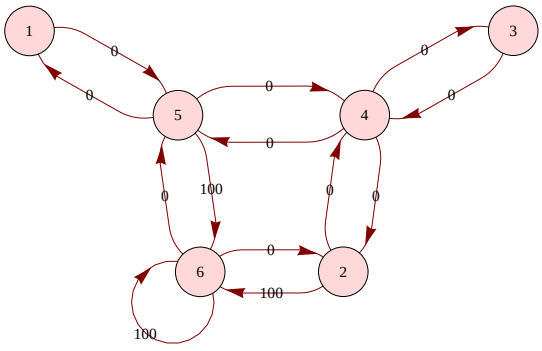


Slika 1: *Agent se nalazi u jednoj od soba, mora izaći van*

Opis algoritma Q learning i primjer

Zamislimo da imamo robota kojeg moramo naučiti kako da se kreće po nekom prostoru npr. stanu koji se sastoji od nekoliko soba i vrata koja ih povezuju. Ako se s početne točke počne kretati u proizvoljnom smjeru, velika je vjerojatnost da će doći do zida i neće moći dalje. To se može smatrati nepoželjnijim stanjem nego prolazak kroz vrata u drugu sobu. Učenje bi se u tom slučaju odvijalo tako da se za svaki sudar sa zidom daje minimalna nagrada, a za prolazak kroz vrata veća, a za stizanje na cilj najveća nagrada. Pogledajmo ipak jednostavniji primjer na slici 1; prikazan je tlocrt stana koji se sastoji od 5 soba i 6 vrata, a zadatak agenta je izaći van.

Isti problem je prikazan dijagramom na slici 2. Stanja su prikazana krugovima, a akcije koje agenta vode iz stanja u stanje strelicama s težinama. Inicijalno su sve težine postavljene na 0, osim za one akcije koje agenta vode u ciljno stanje, one imaju težinu jednaku 100.



Slika 2: *Dijagram stanja prethodnog tlocrta*

U algoritmu se koriste dvije $n \times n$ matrice; matricu okoliša R i matricu učenja Q. Matrica R predstavlja dijagram stanja. Ako su stanja i i j susjedna (povezana), u matrici će na mjestu (i, j) pisati 0 (ili u modificiranoj verziji programa, težina $\omega \in \mathbb{R}$), ako se iz stanja i prelazi u stanje j koje je ciljno, u matrici

na mjestu (i, j) pisat će 100, a ako dva stanja nisu susjedna, vrijednost u matrici bit će $-\inf$. Matrica Q na početku sadrži 0 na svim mjestima.

Pseudokod učenja funkcije Q

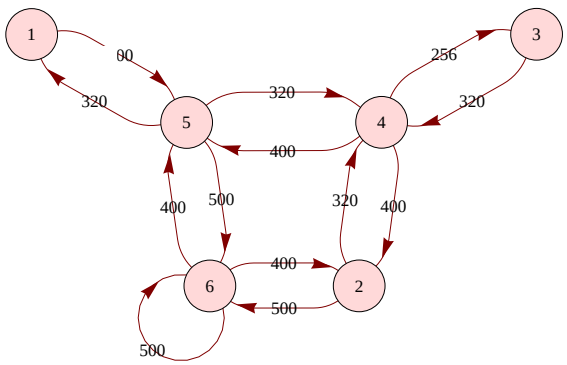
učitaj parametar γ i matricu R
inicijaliziraj vrijednosti matrice Q na 0
while nema konvergencije **do**
  na slučajan način izaberi inicijalno stanje
  while nismo u završnom stanju **do**
    izaberi jedno od mogućih akcija za trenutno stanje
    $Q_{s,a} = R_{s,a} + \gamma \cdot \max_i\{Q_{a,a_i}\}$
    postavi sljedeće stanje za trenutno stanje
  end while
end while

Pomoću ovog algoritma agent uči iz svog iskustva tj. treninga. U svakoj epizodi agent istražuje svoj okoliš (matrica R) i (ne) dobiva nagradu ako (ne) dođe do cilja. Što dulje uči (tj. što više epizoda izvede), bolje će upoznati svoj okoliš i više naučiti (tj. naći će točniju matricu Q). Parametar γ je unaprijed izabrana vrijednost između 0 i 1. Više o značenju parametra γ kasnije je objašnjeno uz sliku 6.

Kad je Q matrica završena, iz nje je lako pročitati slijed akcija koji daje najbrži put od nekog početnog do ciljnog stanja.

Pseudokod nalaženja najkraćeg puta

učitaj matricu Q i početno stanje
while trenutno stanje != finalno stanje **do**
  za trenutno stanje nađi akciju koja ima najveću vrijednost u matrici Q
  za trenutno stanje odaberi ono stanje u koje vodi prethodno odabrana akcija
end while



Slika 3: *Dijagram stanja iz perspektive funkcije Q*

Implementacija

Q learning smo implementirali u programskom jeziku python. Zašto python? Jezik python stvoren je s ciljem lake čitljivosti, velike snage s jednostavnom sintaksa, što omogućuje brzo prototipiranje. Cilj ovog projekta nije bio napraviti jako brzi program, nego ispitati skalabilnost algoritma. Dodatne prednosti pythona su mogućnost integracije sa matematičkim softverom SAGE te vrlo dobro skaliranje sa sve boljim poznavanjem problema.

Generiranje grafova

Kreiranje algoritam za slučajno generiranje problema - grafova je jednako (ako ne i više) težak problem od samog Q learning algoritma. To što komplicira problem su uvjeti koje traženi graf mora zadovoljavati:

- Mora biti povezan.
- Ne smije biti *pregust*.
- Na slučajan način generiramo nagrade.

Da bismo mogli bolje analizirati dobivene rezultate, stvorili smo dvije vrste grafova, nazvali smo ih *gusti* i *dugi*.

U gustim grafovima čvorovi su povezani s relativno malo drugih čvorova. Većina veza je između čvorova koji su *blizu* te čvor može biti povezan s najviše jednim čvorom koji mu nije blizu.

U dugim grafovima čvorovi su povezani s nekoliko *najbližih* čvorova, dakle sve veze su između čvorova koji su *blizu*.

Korištenje programa

Program se sastoji od sljedećih funkcija:

```
Generate(size, cilj, tip='gusta',
raspon=(0, 0))
```

- *size* - broj stanja u dijagramu
- *cilj* - index ciljnog čvora, brojeći od 0. Nagrada je automatski postavljena na 100.
- *tip* - tip matrice koja se generira. Default vrijednost je 'gusta' - stanja mogu biti povezana sa svim stanjima, ali je mala vjerojatnost da bude povezana sa stanjima koja su daleko. Druga mogućnost je 'duga' - stanja mogu biti povezana samo sa nekoliko najbližih stanja
- *raspon* - uređen par cijelih brojeva (a, b), vrijednosti matrice a su iz segmenta $[a, b]$

Funkcija vraća matricu nagrada sljedećeg formata:

- - ako stanja x i y nisu spojena, $m(x, y) = -inf$
- - ako jesu spojena, $m(x, y) = z \in [a, b]$

`ToString(matrica)`
Sprema matricu u jedan string, redovi su odvojeni znakom '\n', a elementi reda zarezom.

`AdjacencyMatrix(matrica)`
Iz matrice generira matricu susjedstva koja se može koristiti za crtanje grafova softverom kao što je SAGE ili Mathematica.

`Parse(cijeliFajl)`
Iz stringa koji sadrži informacije o dijagramu stanja, koji je spremljen kao niz vrijednosti odvojenih zarezom, a redovi su odvojeni znakom '\n', generira se matrica koja se može koristiti za učenje.

`Imp(x, n=7)`
Improved Matrix Print - Ispisuje matricu u oku ugodnom formatu.

`LearnQ(R, gamma, kraj, bench=False)`
Vraća matricu Q. Pri tomu je:

- R - matrica nagrada,
- *gamma* - koeficijent,
- *kraj* - ciljni čvor, brojeći od 0,
- *bench* - False (default) ili True - uz matricu Q vraća i broj izvršenih epizoda i broj koraka algoritma kao uređenu trojku

`FindPath(Q, poc, kraj)`
Vraća listu čvorova koji tvore najkraći put od čvora *poc* do čvora *kraj*, prema matrici Q.

Budući da je program napisan u pythonu, jedini preduvjet za njegovo izvršavanje je instaliran python na računalu. Nadalje, program nije kompiliran u strojni kod nego se prilikom pokretanja interpretira, pa program mora biti svjestan putanje to python interpretera. Pretpostavljena lokacija je `/usr/bin/python`, ukoliko to nije slučaj na računalu na kojem se želi pokrenuti program, putanju treba podesiti u prvoj liniji programa ili program pokrenuti direktno iz pythona.

Sam program se može koristiti na dva načina. Prvi način je direktno pokretanje na danom problemu. Ako primjerice imamo u datoteci `R.mat` zapisanu matricu R i zanima nas najkraći put od čvora 0 do ciljnog čvora 5, to možemo dobiti sa:

```
qlearning.py R.mat 0 5
```

ili:

```
python qlearning.py R.mat 0 5
```

Put se ispisuje na standardni izlaz. Npr:

```
0 -> 4 -> 5
```

Drugi način korištenja programa je kao *modul*, tj. korisnik može sam napisati program koristeći funkcije koje su definirane u programu. Primjer programa koji oponaša defaultni rad našeg programa bi bio:

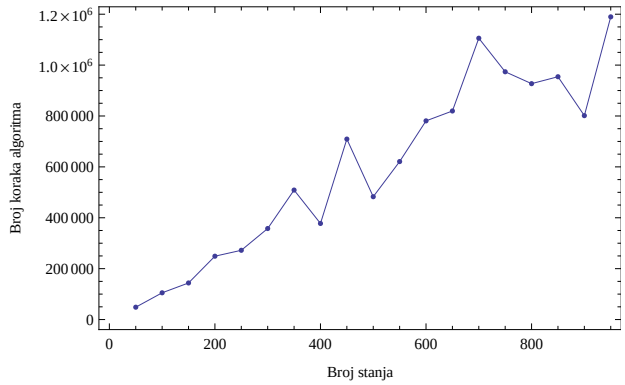
```
import sys
import qlearning as QL
filename = sys.argv[1]
poc = int(sys.argv[2])
kraj = int(sys.argv[3])
f = open(filename, 'rU')
cijeliFajl = f.read()
R = QL.Parse(cijeliFajl)
Q = QL.LearnQ(R, 0.8, kraj)
P = QL.FindPath(Q, poc, kraj)
print ' -> '.join(map(str, P))
```

Rezultati i zaključak

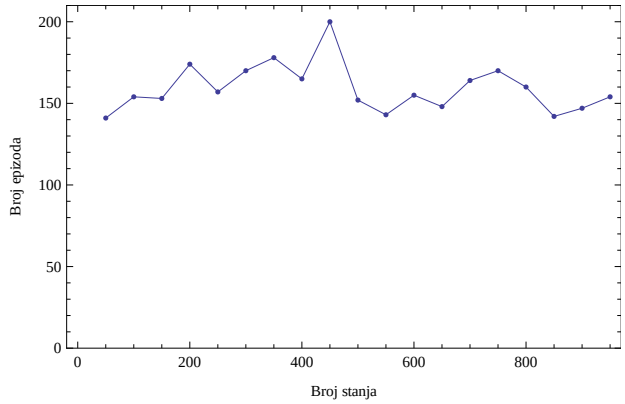
Na slici 4 vidimo odnos broja koraka algoritma s obzirom na broj stanja. Taj odnos je očekivano približno linearan. Mjerenja su izvršena tako da se za svaki fiksirani broj stanja generiralo 10 grafova i onda se uzela aritmetička sredina broja koraka od tih 10 slučajeva. Ovdje treba napomenuti da ti brojevi dosta variraju jer se počeci epizoda algoritma biraju na slučajan način, pa se obzirom na odabir, algoritam može završiti i puno brže, ali i puno sporije. To je također razlog zašto ne možemo analizom algoritma (brojanjem koraka) odrediti njegovu složenost.

Slika 5 nam pokazuje broj epizoda u odnosu na broj stanja. Vrijednost matrice Q na ciljnom elementu konvergira u $100/(1-\gamma)$. Pa kako se γ približava 1, ta vrijednost raste i samim tim je potrebno više epizoda da se dostigne.

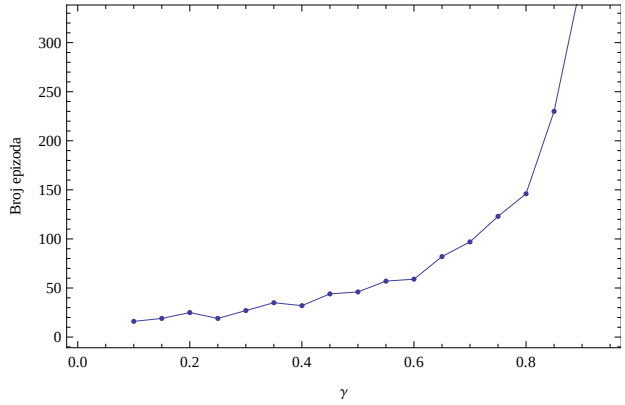
Slika 6 pokazuje broj epizoda u odnosu na vrijednost γ. Ako je vrijednost parametra γ bliža 0 agent više preferira veliku nagradu odmah nakon izbora svake akcije, a ako je bliža 1, agent je spreman pričekati da bi kasnije dobio veću nagradu. Kad bi γ bila jednaka 1, matrica Q ne bi konvergirala.



Slika 4: *Broj koraka algoritma u odnosu na broj stanja*



Slika 5: *Broj epizoda u odnosu na broj stanja*



Slika 6: *Broj epizoda u odnosu na vrijednost* γ

Izvori

- *Q-Learning By Examples*, Kardi Teknomo, `people.revoledu.com/kardi/tutorial/ReinforcementLearning`
- *Q-learning*, Wikipedia, `en.wikipedia.org/wiki/Q-learning`
- *Reinforcement Learning Toolbox*, `www.igi.tugraz.at/ril-toolbox/downloads/index.html`