Maksimizacija nagrade pri problemu nestacionarnega večrokega bandita

Tom Vodopivec
Univerza v Ljubljani, Fakulteta za računalništvo in informatiko

24. 11. 2014

Uvod

Ta dokument je priloga k rešitvi Celtrinega programerskega izziva Jackpot. V nadaljevanju je kratek opis izziva, navodila za uporabo programske kode in opis uporabljenih metod.

Celtrin programerski izziv Jackpot

Naloga tekmovalcev je bila razviti algoritem, ki čim bolj optimalno reši t.i. problem večrokega bandita (ang. Multi-armed Bandit, http://en.wikipedia.org/wiki/Multi-armed_bandit). V omejenem številu potegov igralnih avtomatov je potrebno čim več nagrad, pri čemer so verjetnostne porazdelitve avtomatov neznane in medsebojno neodvisne. Dodatno težavo pri Celtrinem izzivu predstavlja nestacionarnost avtomatov, slednji namreč spreminjajo verjetnost nagrade skozi čas. Problem se prevede na iskanje optimalnega ravnovesja med raziskovanjem prostora in izkoriščanjem znanja, ki je dobro raziskan na področju spodbujevalnega učenja (ang. Reinforcement Learning, http://en.wikipedia.org/wiki/Reinforcement_learning).

Celtra je objavila podrobna navodila za izziv na naslovu http://celtra-jackpot.com/.

Izvorna koda in navodila za zagon

Rešitev je implementirana v programskem jeziku Python 2.7.8. Izvorna koda je objavljena na naslovu https://github.com/hruska2/Celtra-Jackpot. V mapi »dist« se nahaja zagonska datoteka »Jackpot.exe«, ki kot edini parameter sprejme naslov strežnika skupaj s številko primera v obliki *http://<ime_domene>/<številka_primera>.* Primer zagona za testni primer 10 je:

```
Jackpot.exe http://celtra-jackpot.com/10
```

Primer zagonske skripte je podan v datoteki »Execution example.bat« v isti mapi. Za zagon je potreben operacijski sistem Windows in programski paket Python 2.7.x (https://www.python.org/download/releases/2.7.8/).

Zagon izvorne kode s tolmačem za Python se izvede iz datoteke »Jackpot.py«. Po želji se lahko kodo prevede v zagonsko datoteko za Windows okolje iz datoteke »setup.py« z razširitvijo py2exe (http://www.py2exe.org/). Izvajanje prevedene kode je približno 10-krat hitrejše od tolmačenja.

Opozorilo: istočasno se lahko poganja samo ena instanca (omejitev zaradi uporabljenih knjižnic – ob koncu izvajanja posamezne instance se prekine povezava s strežnikom za vse instance).

Ocena vloženega časa: 80 ur

Metodologija

Z eksperimentiranjem sem ugotavljal učinkovitost naslednjih strategij za reševanje problema večrokega bandita: naključna strategija, *epsilon-greedy*, *softmax*, *UCB1*, *UCB-tuned*, *POKER* in nekatere izpeljanke navedenih [1]. Evalvacijo in optimizacijo učnih parametrov sem opravljal križno na množici 40-ih primerov. Lastnosti testnih primerov so bile takšne, kot jih je določila Celtra: 2-10 avtomatov, 500-30000 potegov. Učinkovitost algoritmov sem meril kot relativno povečanje prejete nagrade v primerjavi z uporabo naključne strategije.

Strategija *UCB-tuned* [2] se izkaže kot najboljša izbira za nestacionarne avtomate in tudi v splošnem. Sicer pa strategija *POKER* [3] dosega še boljše rezultate na stacionarnih avtomatih, toda v osnovni različici deluje zelo slabo na nestacionarnih avtomatih, kar bistveno zmanjša njeno uporabnost.

Uvedel sem približno avtomatsko razpoznavo stacionarnosti problemske domene (posameznega primera) z dodatnim algoritmom *UCB-tuned*, ki ob vsakem potegu ocenjuje ali bi potegnil avtomat, ki ga predlaga metoda *UCB-tuned*, ali bi potegnil tistega, ki ga predlaga *POKER*. Končni rezultat je algoritem, ki delno združi prednosti obeh strategij in s tem bistveno zviša splošno učinkovitost.

Kot dodatni mehanizem sem uvedel še detekcijo točke spremembe verjetnostne porazdelitve posameznega avtomata po statistični metodi Page-Hinkley [4]. S tem se poveča učinkovitost algoritma *POKER* na nestacionarnih avtomatih.

Nazadnje sem še delno optimiziral parametre uporabljenih metod. Težo raziskovanja prostora stanj pri strategiji *UCB-tuned* sem definiral kot linearno funkcijo v odvisnosti od števila igralnih avtomatov in največjega dovoljenega števila potegov v posameznem primeru.

Reference:

[1] Kuleshov V., Precup D., Algorithms for the Multi-armed Bandit Problem, 2000. Dostopno na: http://www.cs.mcgill.ca/~vkules/bandits.pdf.

[2] Auer et al., Finite-time Analysis of the Multiarmed Bandit Problem, 2002. Dostopno na: http://homes.di.unimi.it/~cesabian/Pubblicazioni/ml-02.pdf.

[3] Vermorel J., Mohri M., Multi-Armed Bandit Algorithms and Empirical Evaluation, 2005. Dostopno na: http://www.cs.nyu.edu/~mohri/pub/bandit.pdf

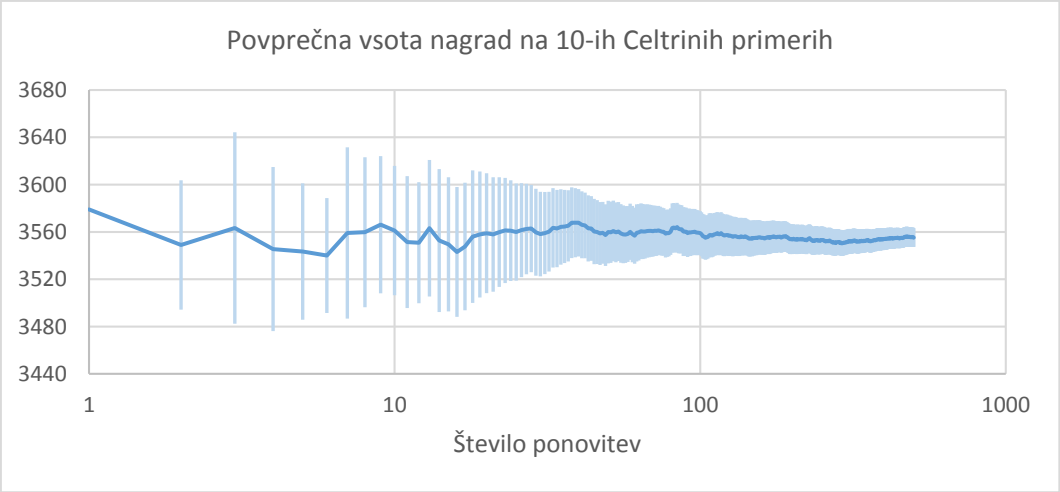
[4] Hartland et al., Change Point Detection and Meta-Bandits for Online Learning in Dynamic Environments, 2007. Dostopno na: https://hal.archives-ouvertes.fr/file/index/docid/164033/filename/cap07.pdf.

Rezultati

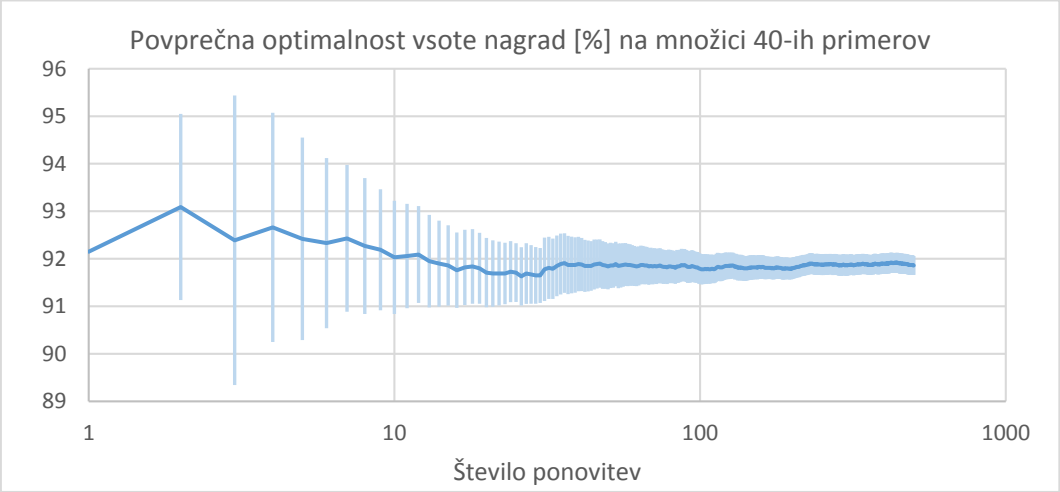
Teoretično optimalna skupna vsota nagrad na 10-ih Celtrinih primerih znaša 3919. Naključna strategija doseže 2805. Razviti algoritem doseže 3555 ± 8 pri 500 ponovitvah in 99% intervalu zaupanja, kot je razvidno iz slike 1. Optimalnost algoritma je približno 90.7%. Izgubo (ang. Regret) je za približno 67.4% manjša kot pri naključni strategiji.

Opozorilo: pri manjšem številu ponovitev so odstopanja lahko bistveno večja zaradi visoke naključnosti avtomatov. Na primer, pri 10 ponovitva V primerjavi z naključno strategijo zmanjša h je odstopanje ± 55. Predlagam, da Celtra izvede več kot 10 ponovitev najboljših tekmovalnih algoritmov, če se izkaže, da so razlike med njimi majhne.

Kot zanimivost prilagam tudi graf učinkovitosti algoritma na množici 40-ih testnih primerov (slika 2). Po 500 ponovitvah je izmerjena optimalnost algoritma 91.9% ± 0.2%, medtem ko je pri 10 ponovitvah odstopanje ± 1.2%.



Slika 1. Učinkovitost razvitega algoritma na 10-ih Celtrinih testnih primerih. Temno modra črta ponazarja izmerjeno povprečno vsoto prejetih nagrad, svetlo modre črte pa 99% interval zaupanja.



Slika 2. Učinkovitost razvitega algoritma na 40-ih testnih primerih.