Univerzitet u Novom Sadu, Fakultet tehničkih nauka

OAS Informacioni inženjering

Monte Carlo Tree Search i njegove primene

Seminarski rad iz predmeta

OSNOVI RAČUNARSKE INTELIGENCIJE

Autori

Nikola Matijević, IN2-2018

Marko Todorčević, IN3-2018

Novi Sad, Godina 2020

Sadržaj

1. Uvod

Seminarski red obuhvata temu *Monte Carlo Tree Search* kraće MCTS, sa kratkim upoznavanjem sa opštijom *Monte Carlo* metodom kao I sa *Reinforcement Learning* paradigmom.

MCTS je heuristički algoritam pretraživanja koji je kombinacija klasičnih implementacija pretraživanja stabla i može da se kombinuje sa principima mašinskog učenja kao što je *reinforcement learning*. Korišćenjem UCT (*Upper Confidence Bound* 1 za stabla) ostvaren je revolucionaran korak za kreiranje agenata koji igraju određene tipove igrara prilikom ostvarenja MCTS algoritama.

Algoritmi pretraživanja stabla povezuju potencijalna stanja datog problema radnjama koje će ih postići, a mi zatim biramo algoritme koji mogu pretraživati unutar ovog prostora kako bi doneli inteligentne odluke koje daju željeni ishod istraživanjem stabla kako bi se pronašlo željeno ciljno stanje. Dok je *reinforcement learning* grana algoritama mašinskog učenja koja uči dobre strategije za problem koji pokušava da reši. Cilj mu je postići optimalnu strategiju ponovnim korišćenjem najboljih radnji koje su pronađene u tom problemu, međutim uvek postoji mogućnost da trenutna akcija u stvari nije optimalna kao takva. Uzimajući u obzir da odabrana radnja ne mora biti optimalna, algoritam periodično vrednuje i druge mogućnosti.

MCTS u suštini spaja ove dve prakse zajedno, koristi model stabla sveta, a zatim traži najbolji mogući put istražujući određene pododelke stabla računajući koliko su bile efikasne, i ažurira vrednost stanja. Radeći ovo u velikom broju iteracija, on može uspostaviti puki put kojim će proći kroz stablo i dati najbolji odgovor na dati problem.

2. *Monte Carlo* metod

Monte Carlo metode, zapravo, čini grupa računarskih algoritama koji se oslanjaju na ponavljanje slučajnih pokušaja da bi se dobili numerički rezultati. Često se koriste u rešavanju fizičkih i matematičkih problema i veoma su korisni u slučajevima kada je nemoguće koristiti druge matematičke metode. Ove metode se najčešće koriste u tri klase slučaja: optimizaciji, numeričkoj integraciji i generisanju uzoraka kod raspodele verovatnoće.

Monte Carlo metode variraju, ali teže da isprate određeni obrazac:

1. Definiše se domen ulaza,
2. Generiše se niz slučajnih ulaza pomoću raspodele verovatnoće,
3. Izvrše se deterministički proračuni nad ulazima i
4. Izvrši se sažimanje rezultata.

Monte Carlo metod, koji se oslanja na stohastički način generisanja numeričkih vrednosti za determinističke probleme koji su teški ili čak nerešivi primenjivanjem drugih metoda i pristupa rešavanju istih, pojavljuje se čak 1940. godina.

U svojoj doktorskoj tezi, 1987. *Bruce Abramson* spaja *minimax search* sa modelom očekivanog ishoda zasnovanog na stohastičkim potezima umesto uobičajene statičke funkcije procene. Došao je do zaključka da se model očekivanog ishoda ponaša precizno, lako procenjivo i nezavisno od domena istraživanja. Eksperimentisao je detaljno sa igrom Iks-oks i kasnije sa mašinski generisanim funkcijama procene za Reversi i Šah.

Takve metode su dalje istraživane i uspešno primenjene u heurističkim pretragama u polju automatizovanog dokazivanja teorema od strane *W. Ertel*, *J. Schumann*, i *C. Suttner* 1989. godine, i samim tim unapredili su tadašnje eksponelncijalno vreme pretraživanja neinformisanih algoritama pretrage kao što su *breadth-first search* (BFS), *depth-first search* (DFS) ili *iterative deepening* (ID). Vreme izvršavanja BFS algoritma je reda $O(|V|+|E|)$, gde je $|V|$ ukupan broj čvorova u stablu, a $|E|$ ukupan broj grana koje povezuju čvorove. Na prvi pogled ovo i ne izgleda kao da ima eksponencijalno vreme izvršavanja, ali obzirom na visok faktor grananja u igrama spomenitum gore, zaista primećujemo eksponencijalan rast. Kod DFS i ID algoritama to je malo očiglednije, ukoliko pretopstavimo vreme prisutpa čvoru da iznosi $O(1)$, sa faktorom grananja b i maksimalnom dubinom m, u najgorem slučaju imamo $1+b+b^2+...+b^{m-1}$ čvorova. Pojednostavljenjem ovoga u $(b^m-1)/(b-1)$ dobijamo eksponencijalno vreme izvršavanja $O(b^m)$.

1992. godine *B. Brügmann* prvi put primenjuje ovu ideju u polju *Computer Go*, podoblasti veštačke inteligencije koja se bavi specifično kreiranjem programa koji „igraju" društevnu igru Go. Ne dugo nakon toga, Chang i saradnici predlažu ideju rekurzivnog kretanja i *backtracking*-a sa prilagodljivim izborima uzorkovanja u njihovom algoritmu *Adaptive Multi-stage Sampling* (AMS) za model Markovljevih procesa odlučivanja. AMS je bio prvi rad koji je istraživao ideju *exlporation-exploitation* zasnovanih na UCB (*Upper Confidence Bound*) u konstruisanju uzorkovanih, odnosno simuliranih (Monte Carlo) stabala, i bio je glavna ideja nad kojom su zasnovani UCT (*Upper Confidence Trees,* vise u nastavku).

Na primer, uzimimo u obzir krug upisan u kvadrat. S obzirom na to da je odnos ovih površina π/4, vrednost π se može aproksimirati upotrebom *Monte Carlo* metode:
1. Nacrtati kvadrat, zatim upisati četvrtinu kruga u isti
2. Ravnomerno prosuti neke objekte (zrna peska ili pirinča, na primer) jednake veličine preko kvadrata
3. Prebrojati ukupan broj zrna unutar kvadranta kruga
4. Odnos između prebrojanog broja zrna i ukupnog broja je procena odnosa dve površine, π/4. Rezultat se onda pomnoži sa 4, čime se dobije broj π.

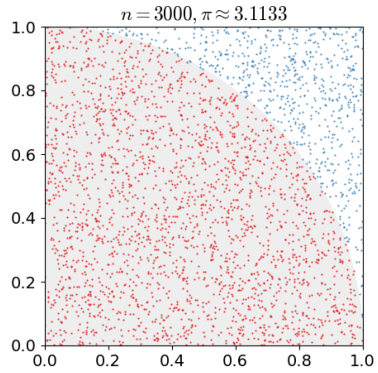
U ovoj proceduri domen ulaza je kvadrat u koji je upisan kvadrant kruga. Generišemo broj ulaza prosipanjem preko kvadrata, zatim izvršimo proračune na svaki ulaz (pitamo se da li je zrno palo u krug). Napokon, sažimamo rezultate da bi dobili konačan rezultat, aproksimaciju π. Ovde imamo dve bitne napomene: prvo, ako zrna nisu ravnomerno raspodeljena, onda je naša aproksimacija nepotpuna. Drugo, mora da postoji veliki broj ulaza. Aproksimacija je generalno loša samo ako je par zrna palo u ceo kvadrat. U proseku kvalitet procene se povećava sa većim brojem zrna u kvadratu. Upotreba *Monte Carlo* metode zahteva veliku količinu brojeva, što je prouzrokovalo formiranje generatora pseudobrojeva čijom upotrebom je proračunavanje olakšano.

In [1]:
```
import numpy as np

n = 1000000

x = np.random.uniform(0, 1, n)
y = np.random.uniform(0, 1, n)

tacaka_u_kruznici = 0

for i in range(n):
    if(x[i] ** 2 + y[i] ** 2 < 1):
        tacaka_u_kruznici += 1

pi = 4 * tacaka_u_kruznici / n
pi
```

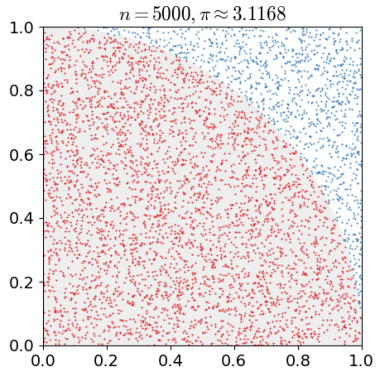
Out[1]: 3.141424

Slika 1: Algoritam opisan iznad

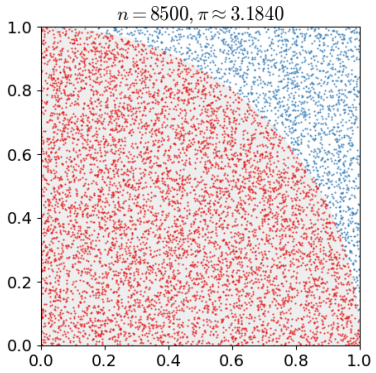
Monte Carlo metoda primenjena na aproksimaciji vrednosti π na slikama 1-6. Nakon ucrtavanja 30.000 proizvoljnih tačaka, procena π je u 0,07% od prave vrednosti.



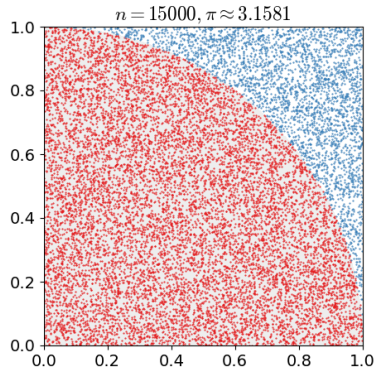
Slika 2. 3.000 ucrtanih tačaka



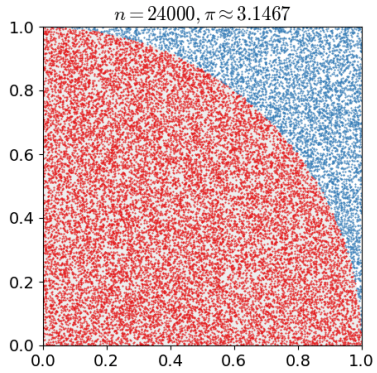
Slika 3. 5.000 ucrtanih tačaka



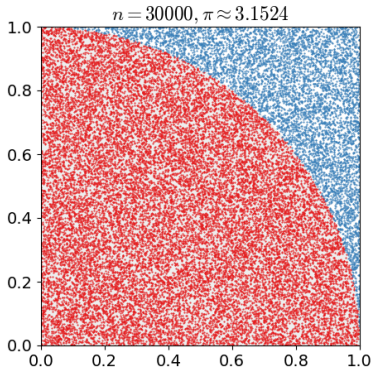
Slika 4. 8.500 ucrtanih tačaka



Slika 5. 15.000 ucrtanih tačaka



Slika 6. 24.000 ucrtanih tačaka



Slika 7. 30.000 ucrtanih tačaka

3. *Monte Carlo Tree Search*

2006. godine, inspirisani radovima njihovim prethodnika, *Remi Coulom* opisuje primene *Monte Carlo* metode u igricama zasnovanim na pretragama stabla, i daje im naziv *Monte Carlo Tree Search* (MCTS), *L. Kocsis* i *Cs. Szepesvari* razvijaju UCT algoritam, i *S. Gellz* i saradnici implementiraju UCT algoritam u njihov MoGo program. 2008. godine, MoGo postiže *dan* (master) nivo u igri Go, na tabli 9x9, dok je program Fuego počeo da pobeđuje protiv jakih amaterskih igrača na istoj tabli.

Januara 2012. godine, program Zen pobeđuje 3:1, na punoj, 19x19 tabli protiv amatera *2-dan* nivoa. *Google Deepmind* razvija program *AlphaGo*, koji u oktobru 2015. godine postaje prvi *Computer Go* program koji je pobedio profesionalnog Go igrača, i to bez ikakvih ograničenja i na punoj 19x19 tabli. U martu 2016. godine *AplhaGo* postiže počasni *9-dan* nivo na 19x19 tabli pobedivši *Lee Sedol*-a u partiji od 5 igara (4:1). *AlphaGo* predstavlja značajan napredak u odnosu na prethodne Go programe kao i posignuće u mašinskom učenju kako koristi MCTS zajedno sa veštačkim neuronskim mrežama za biranje poteza kao i dodeljivanje vrednosti potezima.

MCTS algoritam se takođe koristi u programima koji igraju druge drustvene igre, kao što su: *Hex*, *Havannah*, *Game of the Amazons* i *Arimaa*, u igricama koje se igraju u realnom vremenu, kao što su: *Ms. Pac-Man* i *Fable Legends*, kao i u nedeterminističkim igrama kao što su: skat, poker, *Magic: The Gathering* i *Settlers of Catan*.

Generalno, MCTS algoritmi koriste stroge (heuristike) metode pretraživanja radi identifikovanja najbolje dostupne akcije u datoj situaciji. Oni stiču znanje simuliranjem datog problem i na taj način zahtevaju bar njegov generativni model (poznat i kao *forward, simulation or sample* model), koji može biti jednostavniji od kompletnog modela zadatog problema. MCTS algoritmi postepeno grade asimetrično stablo pretraživanja koje se vodi u pravcu koji najviše obećava.

Motivacija je bila želja da se kreiraju programi koji igraju igre, optimalno u razumnom vremenu. Postoje četiri tipa igara:

- deterministčke igre sa potpunim informacijama (šah, *Go*),
- deterministčke igre sa nepotpunim informacijama (podmornice),
- nedeterministčke igre sa potpunim informacijama (monopol) i
- nedeterministčke igre sa nepotpunim informacijama (igre sa kartama).

Deterministčke igre su igre u kojima nema elementa šanse, dok nedeterministčke igre sadrže taj element. Igre sa potpunim informacijama podrazumevaju da svaki igrač u svakom trenutku vremena ima pristup informacijama o stanju drugih igrača, kao i o stanje igre.

Preteče MCTS algoritma su bili algoritmi sa savršenim informacijama, algoritmi koji su poznavali sva stanja koja su neophodna, u kojima nije bilo mesta za verovatnoćom i oni su kreirali stabla odlučivanja sa svim mogućnostima. Poznatiji primeri tih algoritama su *minimax*, *Pruning* kao i kombinacija prethodna dva *alpha-beta pruning*. Kreiranje ovakvih stabala nije predstavljalo problem za neke jednostavnije igre poput šaha koji se igra na tabli

8x8, gde faktor grananja samog stabla nije previše velik, ali za igru kao što je Go koja se igra na tabli 19x19 faktor grananja dostiže broj 2^{250} , formiranje stabla odlučivanja u ovom slučaju je nezamislivo čak i na današnjim računarima.

3.1. Princip rada

Fokus MCTS-a je na analizi najboljih mogućih poteza, proširivanje stabla pretrage zasnovano na Monte Karlo metodi, gde je domen ulaza prostor pretrage. Primene MCTS-a u igrama je zasnovano na mnogo *playout*-a, drugačije *roll-out.* U svakom *playout*-u, igra se odigra do samog kraja birajući poteze na nasumičan način. Finalni rezultat svakog *playout*-a se onda uzima u obzir prilikom postavljanja novih težina svakog čvora unutar stabla, to omogućava da bolji čvorovi imaju veću šansu da budu izabrani kao potez koji će se zaista i odigrati.

Najosnovniji način korišćenja *playout*-a jeste da se za svaki legalni (dozvoljeni) potez trenutnog igrača primeni isti broj *playout*-a, zatim se bira potez koji je doveo do najviše pobeda. Efikasnost ove metode, koja se naziva i *Pure Monte Carlo Game Search*, često se povećava s vremenom (brojem poteza), jer se više igranja dodeljuje potezima koji su često rezultirali pobedom trenutnog igrača prema prethodnim igrama. Svaka iteracija MCTS-a sadrži četriri koraka:

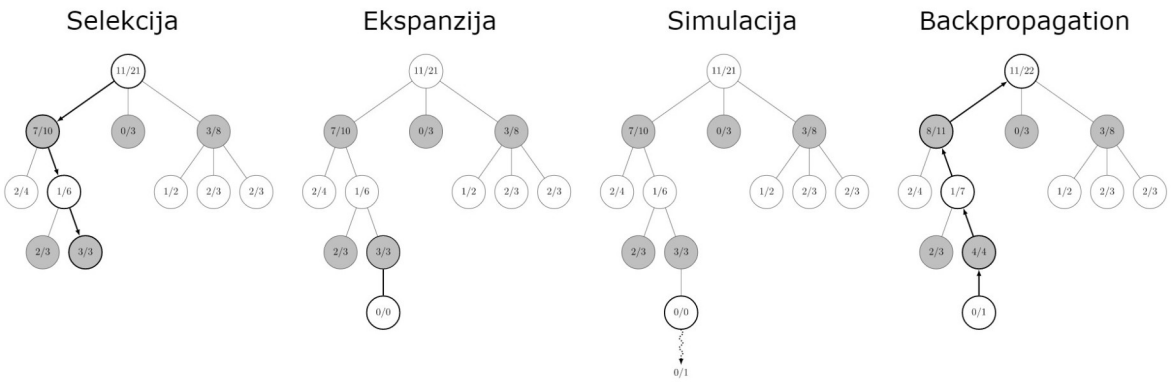
1. Selekcija: počni od korena ***R*** i iterativno selektuj dete čvor na slučajan način sve dok ne stigneš do lista ***L***. Koren predstavlja trenutno stanje igre, dok je list bilo koji čvor koji potencijalno poseduje dete čvor iz kojeg simulacija nije inicirana.

2. Ekspanzija: Ukoliko se u čvoru ***L*** igra ne završava, bilo da je pobeda, poraz ili nerešeno, za bilo kojeg od igrača, kreiraj po jedan čvor za svaki mogući potez i izaberi jedan čvor ***C*** od njih.

3. Simulacija: Izvrši jedan nasumično odabran *playout* iz čvora ***C***. Ovaj korak se drugačije zove i *playout* ili *rollout*. *Playout* može biti toliko jednostavan kao što je odabir uniformno nasumičnog poteza dok se igra ne završi.

4. *Backpropagation*: Koristi rezultat dobijen u *playout*-u i ažuriraj informacije u čvorovima na putanji od ***C*** do ***R***.

Na slici 7. prikazani su koraci u donošenju jedne odluke, svaki od čvorova pokazuje odnos pobeda naspram ukupong broja *playout*-a od te tačke u igri za tog igrača koji čvor reprezentuje. U selekcionom delu, crni igrač igra. Čvor koren pokazuje da postoji 11 pobeda od ukupno 21 *playout*-a za belog. Crni pobeđuje u ostalih 10 slučajeva od ukupnih 21.



Slika 7. Koraci MCTS-a

Ukoliko beli igrač izgubi u simulaciji, svim čvorovima na obeleženom putu treba inkrementovati brojač simulacija, odnosno imenilac, ali, pošto je beli igrač izgubio samo će se crnom igraču inkrementovati brojač pobeda, odnosno brojilac. Ako u suprotnom, beli igrač pobedi, svi čvorovi na putu bi inkrementovali brojač simulacija (imenilac), ali samo bi se čvorovima koji reprezentuju belog igrača inkrementirao brojač pobeda (brojilac). U igrama u kojima je moguće da nema pobednika, odnosno u kojima je moguć nerešen *playout*, brojač simulacija se inkrementuje na isti način, svim čvorovima na putu, dok se brojač pobeda ovoga puta inkrementuje takođe svim čvorovima na tom putu, ali samo za 0.5. Ovo osigurava da se tokom selekcije, izbori svakog igrača kreću u smeru poteza koji najviše obećava, a to komplemetira cilj svakog igrača da maksimizira vrednost njegovog poteza.

Iteracije pretraživanja se ponavljaju dokle god je vreme za potez nepotrošeno. Onda je potez sa najviše odrađenih simulacija i sa najvećim brojem pobeda odabran kao finalni odgovor.

3.2. *Exploration-exploitation*

Glavna poteškoća u fazi selekcije jeste održavanje ravnoteže između *exploitation*-a dobro istraženih poteza sa visokom prosečnom stopom pobede i *exploration*-a malo posećenih poteza. Prvu formula za balansiranje *exploration-exploitation* dileme u igrama, nazvana UCT (*Upper Confidence Bound 1* primenjeno na stabla), uvode *Levente Kocsis* i *Csaba Szepesvari*. UCT se zasniva na formuli UCB1 izvedenoj od *Auer*, *Cesa-Bianchi* i *Fischer* i dokazivo konvergentnog algoritma AMS (*Adaptive Multi-stage Sampling*) koji se prvi put primenjuje na višestepene modele odlučivanja (konkretno, Markov procesi odlučivanja) autorka *Chang*, *Fu*, *Hu* i *Marcus*. *Kocsis* i *Szepesvari* preporučuju da se za spust niz stablo bira čvor koji daje najveću vrednost prema formuli (1).

$$UCT(C)=\frac{w_i}{n_i}+c\sqrt{\frac{\ln N_i}{n_i}} \quad (1)$$

U ovoj formuli (1):

C - posmatrani čvor
w_i - broj pobeda za posmatrani čvor nakon i -tog poteza
n_i - ukupan broj simulacija za posmatrani čvor nakon i -tog poteza
N_i - ukupan broj simulacija roditeljskog čvora nakon i -tog poteza
c - *exploration* parametar, u teoriji $c=\sqrt{2}$

Prvi deo formule odgovara *exploitation*; visok je za poteze koji imaju visok odnos pobeda naspram simulacija (w_i/n_i). Drugi deo odgovara *exploration*-u; visok je u slučajevima u kojima su potezi bili malo puta simulirani.

3.3. Poboljšanja

Predložene su razne modifikacije osnovne metode MCTS kako bi se skratilo vreme pretraživanja. Neki koriste stručno znanje specifično za domen, drugi ne.

MCTS može da koristi proste ili složene *playout*-e. Prosti *playout*-i se sastoje od slučajnih poteza, dok složeni *playout*-i primenjuju različite heuristike da utiču na izbor poteza. Ove heuristike mogu koristiti rezultate prethodnih *playout*-a ili stručna znanja o datoj igri. Na primer, u mnogo programa koji igraju Go, određeni obrasci postavljenih kamenčića utiču na verovatnoću postavljanja novog u blizini. Paradoksalno, igranje suboptimalno ponekad čini da MCTS u celini deluje bolje kasnije. Takođe, *playout*-i su često skraćeni (tj. zaustavljeni pre dostizanja krajnjeg stanja, pobeda, poraz ili nerešeno), time pokrivamo veći broj različitih *playout*-a, a ne ulazimo toliko duboku u jedan konkretan koje svakako ima manje šanse da se ispuni.

Domensko znanje može biti primenjeno pri kreiranju stabla igre kako bi pomoglo u *exploitation* delu nekih podstabala. Jedna od metoda dodeljuje nenula vrednosti umesto nula vrednosti broja pobeda kao i broja odigranih simulacija pri kreiranju dece čvorova, i na taj način veštački povećavamo ili smanjujemo prosek pobeda u odnosu na broj simulacija što kao efekat ima da će se taj čvor češće ili ređe odabirati.

Osnovni MCTS sakuplja dovoljno informacija da pronađe najbolji mogući potez nakon mnogo iteracija; do tad potezi koje igra su u osnovi nasumični. *Exploration* faza može biti skraćena znatno u specifičnim klasama igara koristeći RAVE (*Rapid Action Value Estimation*). U ovim igrama, permutacije sekvenci poteza dovode do istih pozicija. Tipično, to su igre igrane na tabli u kojima potez podrazumeva postavljanje figure ili postavljanje kamenčića na samu tablu. U takvim igrama vrednost svakog poteza je obično pod malim uticajem drugih poteza.

U RAVE-u, za dati čvor N stabla igre, njegova deca čvorovi C_i sadrže ne samo statistiku pobeda playout-a započetim u N nego i statistiku pobeda svih odigranih *playout*-a u čvorovima ispod N, ako sadrže potez i (takođe kada je potez odigran u stablu, između čvora N i *playout*-a). Na ovaj način sadržaj čvorova stabla su pod uticajem ne samo potezima odigranim odmah u trenutnoj poziciji, nego i istim potezima odigranim kasnije.

Pri korišćenju RAVE-a, u selekcionom korak bira se čvor, za koji modifikovana UCT

$$UCT(C)=(1-\beta(n_i,\tilde{n}_i))\frac{w_i}{n_i}+\beta(n_i,\tilde{n}_i)\frac{\tilde{w}_i}{\tilde{n}_i}+c\sqrt{\frac{\ln t}{n_i}}$$

daje najveću vrednost. U ovoj formuli $\tilde{w}_i$ predstavlja broj pobeda *playout*-a koji sadrži potez i, a $\tilde{n}_i$ je broj svih *playout*-a koji sadrže potez i, i funkcija $\beta(n_i,\tilde{n}_i)$ treba da bude blizu 0 i blizu 1 za relativno male i velike vrednosti n_i i $\tilde{n}_i$, redom. Jedna od mnogih formula za $\beta(n_i,\tilde{n}_i)$, koju je predložio *D. Silver*, kaže da u balansiranoj pozicijji funkcija β treba da izgleda:

$$\beta(n_i,\tilde{n}_i)=\frac{\tilde{n}_i}{n_i+\tilde{n}_i+4b^2n_i\tilde{n}_i}$$

gde je b empirijski odabrana konstanta.

MCTS može biti konkurentno pokrenut sa mnogo niti ili procesa. Postoji nekoliko fundamentalno različitih metoda paralelizacije MCTS-a:

- *Leaf* paralelizacija, tj. paralelno izvršavanje mnogih playout-a iz jednog lista stabla igre.
- *Root* paralelizacija, tj. kreiranje nezavisnih stabala igre paralelno i igranje poteza na osnovu grana koje povezuju koren stabla igre i nezavisna podstabla igre.
- *Tree* paralelizacija, tj. paralelno kreiranje jednog stabla igre, gde se podaci od istovremenog pisanja čuvaju korišćenjem jednog, globalnog *mutex*-a, uz pomoć više *mutex*-a, ili sa *non-blocking* sinhronizacijom.

4. *Reinforcement Learning*

Reinforcement Learning (RL) je uspostavljena paradigma za učenje agenata putem iskustva. U RL problemima, agent čini akcije u okruženju i dobija povratnu informaciju u vidu nagrade i dodatnih informacija trenutnog stanja okurženja u kom se nalazi. Cilj agenata jeste da maksimizuje očekivanu nagradu. Pokušava da nauči koje akcije u kojim stanjima dovode do veće nagrade kako bi koristio stečeno znanje da u budućnosti preduzima za njega pogodnije akcije. Agentu se ne govori šta on treba da radi, već je postavljen u situaciju u kojoj delovanjem mora da otkriva najbolje postupke, greši i uči. Problem postaje teži kada nagrade nisu vremenski bliske. Još jedan od izazova RL-a jeste *exploration-exploitation* dilema, gde agent mora da odluči da li da preduzima akcije za koje zna iz prethodnog iskustva da mu donose nagrade ili da istražuje manje poznate akcije u kojima je mozda sačinjena veća nagrada. Generalno, RL problemi mogu da budu rešeni učenjem vrednosnih funkcija i onda korišćenjem istih da se formiraju smernice kojih bi se trebao pridržavati. Metode koje rade na takav način nazivaju se *action-value* metode, primer tih metoda je dinamičko programiranje kao i *Monte Carlo* metode. *Monte Carlo* metode se klasifikuju kao RL metode zasnovane na uzorcima, koje ne zahtevaju model okruženja, a pritom uče it jednostavnih interakcija.

4.1. Veza Reinforcement Learning-a i MCTS-a

Metode učenja i metode planiranja imaju jake veze koje su posmatrane decenijama (*Sutton* i *Barto*, 1998) i takođe su reanalizirane od strane nekoliko istraživača (*Asmuth* i *Littman*, 2011; *Silver*, *Sutton* i *Muller*, 2012; *Sutton* i *Barto*, 2017). Obe paradigme rešavaju slične probleme sa sličnim pristupima, obe procenjuju iste vrednosne funkcije i obe ažuriraju procene inkrementalno. Ključna razlika se nalazi u izvoru iskustva: za metode učenja ono dolazi iz stvarnih interakcija sa okruženjem, dok za metode planiranja ono dolazi iz simuliranih okruženja. Ovo implicira da, ma koji metod učenja može biti iskorišćen umesto metode planiranja, kada se primeni za simulirana okruženja. Što znači, znajući da se MCTS smatra metodom planiranja i pretrage, njegov mehanizam pretrage je uporediv sa RL metodama zasnovanim na uzorkovanju i da je celokupan MCTS uporediv sa RL korišćenim za planiranje. MCTS istaživači su manje zaiteresovani za generično učenje rešavanja problema i više su fokusirani da napadaju određene igre specifičnim pristupima; te igre su najčešće epizodične, ali znanje često nije očuvano između tih epizoda ili između različitih igara. RL istraživači su najčešće fokusirani na univerzalno primenljiva rešenja za šire klase problema i njihovi fundamenti ih udaljavaju od korišćenja ekspertskog znanja u neprincipijalnim načinima (iako ovo može biti veoma efektivno u praksi). Kao posledicu, metode pretrage su generalno fokusirane na kreiranje strogih pravila izbora, najčešće istražujući heuristike, a RL metode su više fokusirane na razvijanje jake tehnike za ažuriranje stečenog znanja, davajući manje značaja pravilima izbora. Primer ovoga je kako MCTS i RL pristupaju eksploraciji. MCTS metode se uzdaju u samo pravilo izbora (UCB pravilo), dok se RL metode uzdaju u primarno postavljene optimistične pretpostavka, koje dalje vode proces selekcije ka manje posećenim delovim prostora stanja. Iako ove razlike postoje, oba pristupa moraju da rade slične zadatke: efikasno biranje akcije, ažuriranje znanja, balansiranje *exploration-exploitation* dileme. Još jedan velik razlog je česta pretpostavka da u RL nemamo model okruženja, RL razume planirane scenarije zasnovane na uzorcima kao

specijalne slučajeve - kao primenu metoda učenja na simulirano okruženja, za koje nam nije potreban model. Svakako, nije nam ni potrebano da imamo kompletan model, dovoljno je da bude sličan. Alternativno, model čak ni ne mora da bude zadat, već metode mogu ga nauče *online*. Dalje, čak i klasične MCTS su koristile model koji nije bio nužno perfektan niti kompletan, mogao je biti izrazito stohastičan ili čak da simulira samo deo okruženja i koji nije imitirao druge agente. Na primer, u igrama sa više igrača, drugi igrači su često bili shvatani kao deo okruženja koji nije lako predvidiv i bili su modelovani odvojeno. I MCTS i RL metode moraju da se suočavaju sa problemima koji se tiču modela. Metode pretraživanja često kreiraju strukture stabla zasnovane na odlukama, zbog česte prirode igara, i uzimaju u obzir mogućnost posmatranja stanja kao bonus koji im omogućava korišćenje tabela transpozicije. Sa druge strane, u RL posmatranje stanja se ne uzima u obzir: RL istraživanja se više fokusiraju na više generične, potpuno povezane grafove stanja i akcija, gde je transpozicija podrazumevano ukodirana zbog reprezentacije. Iako razlike u reprezentacijama postoje, i RL metode i metode pretrage, teže generaliziciji stečenog znanja, kako bi ga prenele u relative delove prostora stanja.

5. Zaključak

Stablo MCTS algoriotma raste asimetrično kako algoritam pronalazi obećavajuća podstabla, time postiže bolje rezultate od klasičnih algoritama u igrama sa velikim stepenom grananja. Vreme izvršavanja MCTS-a je proizvoljno, naravno što se duže izvršava, rešenje koje proizvede će biti bolje. Kreirano stablo je moguće opet koristiti, od poteza protivnika do poteza protivnika, ali naravno to nije obavezno, takođe ne koristi vrednostnu funkciju, algoritmu je dovoljno da poznaje da li je igra završena. Algroitam se može menjati na razne načine, u čvorovima stabla koje kreira moguće je čuvati dodatne informacije, kao i granama. UCT je moguće modifikovati kako god mi želimo, broj igara za svaki playout može da bude daleko veći. MCTS je fleksibilan algoritam čija modifikacije mogu doneti vrhunske rezultate.

MCTS je najviše slave je stekao kao metoda koja se koristila za kreiranje AlphaGo neuronske mreže koja je uspela da pobedi najbolje igrače u igri Go, šah kao i mnogim drugima što pokazuje da računar nadjačava čoveka u njegovoj igri, koje stotine ljudi hiljadama godina istražuju i traže optimalne strategije. Eksploatisanje ovakve računarske snage za pogodne probleme, u ljudskom oku, imaće mogućnost brzog rešavanja spektra problema koje čovek ne može sam da reši. Svaki pomak u računarskim naukama unapređuje civilizaciju algoritam po algoritam.

6. Literatura

[1] On Monte Carlo Tree Search and Reinforcement Learning [Internet],
Tom Vodopivec, Spyridion Samothrakis, Branko Šter,
Sadržaj dostupan na: http://repository.essex.ac.uk/21129/1/live-5507-10333-jair.pdf

[2] Modification of UCT with Patterns in Monte-Carlo Go [Internet],
Sylvain Gelly, Yizao Wang, Rémi Munos, Olivier Teytaud,
Sadržaj dostupan na: https://hal.inria.fr/file/index/docid/117266/filename/MoGoReport.pdf

[3] Biasing Monte-Carlo Simulations through RAVE Values [Internet],
Arpad Rimmel, Fabien Teytaud, Olivier Teytaud,
Sadržaj dostupan na: https://hal.inria.fr/inria-00485555/document

[4] Monte Carlo Methods [Internet],
Malvin H. Kalos, Paula A. Whitlock,
Sadržaj dostupan na:
https://www.deeplearningitalia.com/wp-content/uploads/2018/03/Monte-Carlo-Methods-Kalos-Whitlock.pdf

[5] A tutorial introduction to Monte Carlo Tree Search [Internet],
Michael C. Fu,
Saržaj dostupan na: https://informs-sim.org/wsc20papers/139.pdf