Game tree
Randomized algorithms version

Seminarski rad u okviru kursa

Konstrukcija i analiza algoritama 2

Matematički fakultet

Vojkan Cvijović 1108/17

vojkancvijovic@gmail.com

5. februar 2019.

Contents

Uvod....2
Deterministički algoritam....3
Algoritam sa nasumičnim odabirom....4
Analiza algoritma sa nasumičnim odabirom....5
Test rezultati....6
Literatura....7

Uvod

Stablo igre predstavlja stablo pretrage koje se koristi da bi se ispitali svi mogući potezi, njihovi rezultati, u pokusaju da se nađe najbolje resenje za datu situaciju. Kompletno stablo igre je stablo igre koje kreće od početne konfiguracije i sadrži sve moguće poteze za svaku od pozicija. Neke od osobina:

- Svaki čvor u stablu predstavlja delimicnu postavku igre. U igrama poput šaha ili X-O čvor predstavlja tablu za igru sa trenutnim rasporedom figura ili oznaka.
- Koren stabla predstavlja početnu konfiguraciju. U igri X-O to je prazna 3 x 3 tabela.
- Svaka grana u stablu predstavlja jedan potez koji je odigrao igrač.
- Svaki list je kraj igre, više se ne moze odigrati ni jedan potez. Svakom listu se pridružuje rezultat, da li je prvi igrač pobedio, izgubio ili pak ako igra dozvoljava i nerešen rezultat. Broj listova u kompletnom stablu pretrage je broj mogućih načina na koji igra može da se završi.
- Svako stablo odlikuje i faktor grananja, koji predstavlja prosečan broj dece po čvoru.

Jedan od načina da se nađe najbolji potez u igri je da se pretraži stablo koristeći algoritam minmax ili neku njegovu varijantu. Minmax algoritam je algoritam za izbor poteza pretrazivanjem stabla igre. On za igrača koji je na potezu određuje najbolji mogući potez u datoj situaciji. Vrši ocenjivanje čvorova i uzima najbolji. Kao ocenu čvoru dodeljuje se minimum ocena čvorova-potomaka, ako je u tom čvoru na potezu protivnik, a kao maksimum ocena čvorova potomaka, u suprotnom. U igrama poput X-O stablo nije teško obići, dok je na primer u šahu stablo je preveliko za pretragu i vrši se samo delimična pretraga stabla.

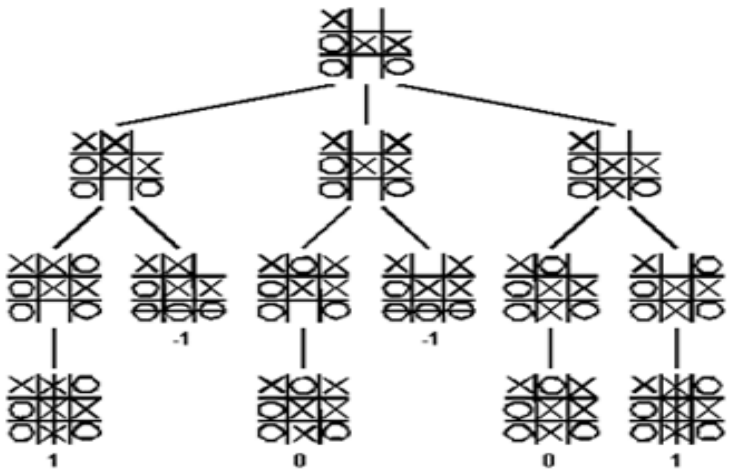

Figure 1 Game tree

Načini za pretragu drveta su:

1. Koristeći deterministički algoritam.
2. Koristeći algoritam sa nasumičnim odabirom.

Deterministički algoritam

Determministički algoritam pokušava da nađe najbolju kombinaciju poteza prolazeći kroz stablo pretrage **redom** dok ne naiđe na cvor u kome pobedjuje ili dok ne završi sa pretragom stabla.

Da bi pojednostavili u primerima ćemo u primerima koristiti igru u kojoj igrač može da pobedi ili izgubi što će biti označeno sa True ili False. Za stablo pretrage ćemo koristiti AND-OR stablo. AND-OR stablo je stablo čiji unutrašnji čvorovi su obelezeni sa “AND” ili “OR” u zavisnosti koju će operaciju primeniti nad svojim potomcima. Listovi AND-OR stabla mogu imati samo True ili False vrednost. Njihova vrednost se rekurzivno propagira na unutrašnje čvorove stabla na način:

- Ako je čvor obeležen sa OR onda on ima vrednost True samo ako neko od njegovih potomaka ima vrednost True.
- Ako je čvor obeležen sa AND onda on ima vrednost True samo ako svi njegovi potomci imaju vrednost True.

Primer:

```
def gt_eval_determ(u):
    '''Returns true if this node evaluates to a win, otherwise false'''
    if u.leaf:
        return u.win
    elif u.op == 'OR':
        for child in u.children:
            if gt_eval_determ(child) == True:
                return True
        return False
    elif u.op == 'AND':
        for child in u.children:
            if gt_eval_determ(child) == False:
                return False
        return True
```

Figure 2 Pseudo kod

Polja čvora u:

- **u.leaf** je True ili False u zavisnosti da li je čvor u list.
- **u.op** polje za operaciju koja se primenjuje na deci čvora u
- **u.win** ima vrednost True ili False u zavisnosti da li je obezbedjena pobeda ili poraz sa postavkom u listu u.
- **u.children** niz dece čvora u kroz koji iteriramo stabla sleva nadesno (od pocetka ka kraju).

Ovaj algoritam radi dobro ali je spor. Bez obzira sto na povoljan ishod moze naići ranije i tako prekinuti izvržšavanje, algoritam u najgorem slučaju mora da prodje kroz sve čvorove, pa je složenost ovog algoritma O(n), gde n predstavlja broj čvorova u drvetu pretrage.

Algoritam sa nasumičnim odabirom

Ključna razlika u odnosu na deterministički algoritam je to što je **nasumičan** obilazak stabla umesto obilaska stabla levo u desno. Važno je napomenuti da nasumični obilazak treba da uključi svu decu čvora, da ne bi neki čvor bio izostavljen i da isključi svu decu koju smo već obišli, da ne bi ponovo izračunavali.

```
def gt_eval_rand(u):
    '''Returns true if this node evaluates to a win, otherwise false'''
    if u.leaf:
        return u.win
    elif u.op == 'OR':
        for child in random_order(u.children):
            if gt_eval_rand(child) == True:
                return True
        return False
    elif u.op == 'AND':
        for child in random_order(u.children):
            if gt_eval_rand(child) == False:
                return False
        return True
```

Figure 3 Pseudo kod

Algoritam ima dve prednosti u odnosu na deterministički algoritam:

- brzinu, u slučaju da je faktor grananja 2.
- praktičnost.

U najgorem mogućem slučaju moramo obići svih n čvorova, te je slozenost u najgorem slucaju O(n), kao i u slucaju deterministickog algoritma. Ali ukoliko je faktor grananja 2 onda algoritam ima očekivano vreme izvršavanja $\theta(n^{0.792})$ što mu daje prednosti u odnosu na deterministički algoritam. Što se tiče praktičnosti, kako algoritam ima nasumičan obilazak stabla to će i potezi izgledati "nasumično" pa postoji mogućnost da zavaramo protivnika. Protivnik neće moći da predvidi obilazak stabla pretrage kako je sam obilazak nasumičan.

Analiza algoritma sa nasumičnim odabirom

Složenost je vrlo jednostavno odrediti. Kako moramo proći kroz sve čvorove u najgorem mogućem slučaju, složenost će biti linearna po broju listova,tj. O(n) gde je n broj listova.

Analiziraćemo očekivano vremene izvršavanja kada se koristi binarno stablo igre.

Trik koji koristimo je da gledamo dva nivoa u stablu. Na primer ako imamo koren kome je pridružena operacija AND i želja nam je da odredimo da li je igrač pobedio, odnosno da li čvor AND vraća vrednost True. Moramo da izračunamo vrednost svakog potomka čvora AND i ne može se koristiti lenjo izračunavanje. Pošto koristimo AND-OR stablo znamo da čvor koji je obeležen sa AND ima potomke koji su obeleženi sa OR. I kako za njih bi želeni da dobijemo vrednost True, zbog operacije OR može da se primeni lenjo izračunavanje i potencijalno se preskoči izračunavanje svakog potomka ukoliko nam jedan od potomaka ima vrednost True. U tom slučaju zbog operacije OR tekućem čvoru možemo da dodelimo vrednost True. Suprotno važi kada je operacija pridružena roditelju OR, želimo da odredimo da li čvor vraća vrednost False.

Moramo da razmatramo dva slučaja, u zavisnosti da li čvor koristi lenjo izračunavaje ili ne. Pa tako definišemo dve rekurentne jednačine. T(n) kao očekivano vreme evaluacije drveta veličine n koje ne koristi lenjo izračunavanje i moraju se izračunati vrednosti svih potomaka. S(n) kao očekivano vreme kada čvor koristi lenjo izračunavanje i potencijalno može se izbeći izračunavanje svih potomaka. Ukoliko gledamo hijerarhiju stabla, čvor koji koristi lenjo izračunavanje ima potomke koji ne koriste lenjo izračunavanje i oni dalje imaju potomke koji koriste lenjo izračunavanje. Iz tog razloga u jednačini T imamo pozive potproblema za S i obrnuto.

Bazni slučaj je isti za oba slučaja

$$T(1)=S(1)=1$$

$$T(n)=1+2*S\left(\frac{n}{2}\right),n\geq 2$$

Broj rekurzivnih poziva u S(n) može da bude 1 ili 2. U slučaju da se algoritmu posrećilo onda će biti dovoljan samo rezultat iz prvog potomka i samim tim ne moramo da računamo vrednost drugog potomka. U suprotnom slučaju mora se računati i vrednost drugog deteta.

$$S(n)=\frac{1}{2}\left(1+T\left(\frac{n}{2}\right)\right)+\frac{1}{2}\left(1+2T\left(\frac{n}{2}\right)\right)$$

Što se pojednostavi

$$S(n)=1+\frac{3}{2}T\left(\frac{n}{2}\right)$$

Kada uključimo S(n) u T(n)

$$T(n)=1+2\left(1+\frac{3}{2}T\left(\frac{n}{4}\right)\right)=3+3T\left(\frac{n}{4}\right)$$

$$T(n)=1+3T\left(\frac{n}{4}\right)=2+9T\left(\frac{n}{16}\right)=3+27T\left(\frac{n}{64}\right)=\ldots=k+3^kT\left(\frac{n}{4^k}\right)$$

Koristimo k = $\log_4 n$

$$n+3^{\log_4 n}T(1)=\Theta(¿)\approx\Theta\left(n^{0.792}\right)$$
$$T(n)=\log_4 ¿$$

Rezultat je značajno poboljšanje u odnosu na deterministični algoritam u slučaju binarnog stabla igre.

Test rezultati

Što se tiče složenosti implementiranog algoritma, u najgorem slučaju moramo da obiđemo sve čvorove stabla igre. Ukoliko se koristi algoritam sa nasumičnim odabirom onda se nasumice promeša redosled obilaska koji se čuva u nizu. Niz je uvek konstantne dužine. Složenost je O(n).

Algoritam ima mogućnost da koristi dve različite funkcije za promenu rasporeda obilaska. Jedna od njih je funkcija shuffle, funkcija iz standardne c++ biblioteke, čije

vreme u velikoj meri zavisi od generatora koji koristi. Pa tako vreme izvršavanja je išlo i do 76 265 751 ns kada se koristi Mersenne Twister 19937 generator, dok je najbolje prosečno vreme zabeleženo sa Ranlux 48 base generatorom 4 639 722 ns. Druga funkcija za promenu rasporeda obilaska ne koristi deljenje prilikom određivanja obilaska pa je zabeleženo bolje vreme. Ona je preuzeta sa bloga[1].

Kako X-O ima koeficijent grananja 4, samim tim nemamo binarno stablo igre i ne možemo odrediti očekivano vreme izvršavanja.

Testiranje je vršeno tako što je 10000 puta ponavljan test u kome se na praznoj tabeli za X-O postavi jedna ozaka da je igrač odigrao, pusti se algoritam, meri mu se vreme, briše se postavljena oznaka sa table i prelazi se na drugo polje. Na kraju se računa prosek.

	Bez promene obilaska[2]	Fast shuffle	shuffle
Prosečno vreme izvršavanja	740 124 ns	909 209 ns	4 639 722ns
Povećanje vremena izvršavanja	/	20.5%	144.971%

Zaključaj je da nam algoritam sa nasumičnim odabirom u ovom slučaju nije pomogao. Ostvareno je usporenje od 20%. Neko potencijalno ubrzanje bi bilo da se implementiraju optimizacije navedene na blogu[3].

Literatura

1. Predrag Janjičić, Mladen Nikolić: Veštačka inteligencija, Beograd, 2018.
2. https://www.usna.edu/Users/cs/roche/courses/s13si486d/u03/
3. https://cs.nyu.edu/courses/spring02/G22.2560-001/andor.html
4. https://www.cs.cmu.edu/~adamchik/15-121/lectures/Game%20Trees/Game%20Trees.html
5. https://lemire.me/blog/2016/06/30/fast-random-shuffling/

1 https://lemire.me/blog/2016/06/30/fast-random-shuffling/
2 Standardni obilazak stabla sa leva na desno, bez primene nekog shuffle algoritma
3 https://lemire.me/blog/2016/06/30/fast-random-shuffling/