RAČUNALNIŠKO IGRANJE IGER S KARTAMI

Mitja Luštrek
Odsek za inteligentne sisteme
Institut Jožef Stefan
Jamova 39, 1000 Ljubljana, Slovenija
Tel: +386 1 4773900; telefaks: +386 1 2519385
E-pošta: mitja.lustrek@bocosoft.com

POVZETEK

Ta članek predstavi algoritme, ki se uporabljajo pri računalniškem igranju iger z nepopolno informacijo (kamor sodi večina iger s kartami). To so predvsem alfa-beta in njegove izpeljanke za preiskovanje drevesa igre ter vzorčenje Monte Carlo za obravnavanje nepopolne informacije. Opiše tudi program za igranje taroka: kako so v njem uporabljeni ti algoritmi in kakšne rešitve sem razvil prav za tarok.

1. UVOD

Igranje iger je področje umetne inteligence, ki je deležno dosti pozornosti, vendar je je večina usmerjena k igram s popolno informacijo, kakršna je npr. šah, igre z nepopolno informacijo, kamor sodi večina iger s kartami, pa so zapostavljene. Glavni razlog je najbrž ta, da je šah verjetno najbolj dognana in ugledna igra, algoritme zanj pa je moč uporabiti tudi pri drugih igrah z popolno informacijo. Drži pa tudi, da je igranje iger z nepopolno informacijo težavnejše, saj nepoznane niso le nasprotnikove poteze, ampak tudi samo stanje igre.

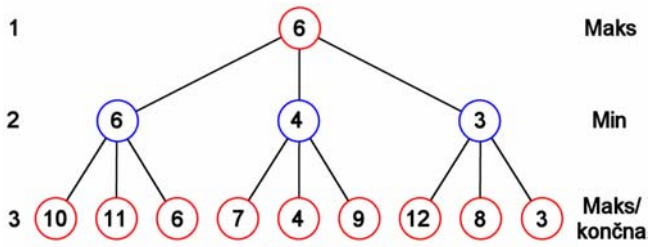
Je pa res, da je osnovna in najpogostejša metoda za računalniško igranje iger – to je preiskovanje drevesa igre – uporabna pri obeh kategorijah. Pri igrah z nepopolno informacijo se navadno uporablja v kombinaciji s kako vrsto simulacije: to pomeni, da se tvorijo nabori manjkajočih podatkov in da se išče na vsakem izmed njih. Obstajajo pa tudi drugačni načini – npr. s planiranjem, a zdi se, da se nobeden ni širše uveljavil.

Metode za računalniško igranje iger sem prikazal na programu za igranje taroka, ki je primerno kompleksna in v Sloveniji precej priljubljena igra. Program, ki se imenuje Silicijasti tarokist, je na voljo na spletni strani **http://tarok.bocosoft.com**. Pri svojem delu sem se oprl predvsem na tehnike, razvite za igranje bridža, ker je bridž od iger, o katerih je na voljo kaj uporabnih informacij, taroku še najbolj podoben. Pa tudi računalniško igranje bridža je od iger z nepopolno informacijo najbolj dognano. Uporabil sem preiskovanje drevesa igre z močno izpopolnjenim algoritmom alfa-beta, nepopolno informacijo pa sem obravnaval z vzorčenjem Monte Carlo.

2. PREISKOVANJE DREVESA IGRE

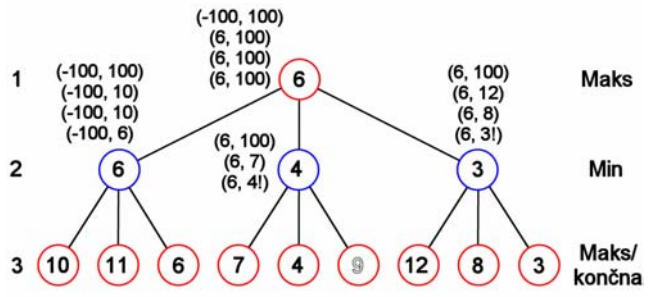
2.1. Minimaks in alfa-beta

Igranje igre lahko predstavimo kot drevo, v katerem so vozlišča stanja igre, povezave pa poteze. Izmenjujejo se plasti, kjer smo na potezi mi, in plasti, kjer je na potezi nasprotnik. Če bi tako drevo razvili do konca, bi lahko natančno predvideli potek igre in vedno izbrali najboljšo potezo. Ker pa je v večini iger celotno drevo preveliko, da bi ga razvili, uporabimo algoritem minimaks [1]. Po njem drevo razvijemo do izbrane globine, nato pa vsak list ocenimo. Ocenjevalna funkcija je odvisna od problema, s katerim se ukvarjamo. Recimo, da si prizadevamo za čim večji rezultat: vozlišča *maks* so potem tista, kjer smo na potezi mi, kajti tam izberemo vejo z največjim rezultatom; v vozliščih *min* je na potezi nasprotnik in v njih izberemo vejo z najmanjšim rezultatom. To ponazarja slika 1.



Slika 1: Drevo igre pri algoritmu minimaks

Očitno pa se v algoritmu minimaks opravi nekaj odvečnega preiskovanja. V primeru na sliki 1 vzemimo, da drevo razvijamo od leve proti desni. V tem primeru bi v vozlišču 4 v plasti 2 lahko *min* pri listu 4 iskanje prekinil, saj bi že



Slika 2: Drevo igre pri algoritmu alfa-beta

vedel, da se v tem vozlišču da doseči vrednost vsaj 4; ker pa je v prejšnjem vozlišču v plasti 2 dosegel vrednost 6, bo *maks* v plasti 1 raje izbral to vozlišče – temu rečemo rez. Algoritem, ki izkorišča to lastnost, se imenuje alfa-beta [2] in je osnova večine današnjih programov za igranje iger (alternative so, a niso dobro raziskane). Ime alfa-beta izvira iz spremenljivk α in β, ki označujeta zgornjo in spodnjo mejo vrednosti, ki jo trenutno lahko dosežemo. Kako bi ga uporabili na primeru s slike 1, kaže slika 2: pri vozliščih so dodani pari (α, β), mesta, kjer pride do reza, pa so označena s klicajem.

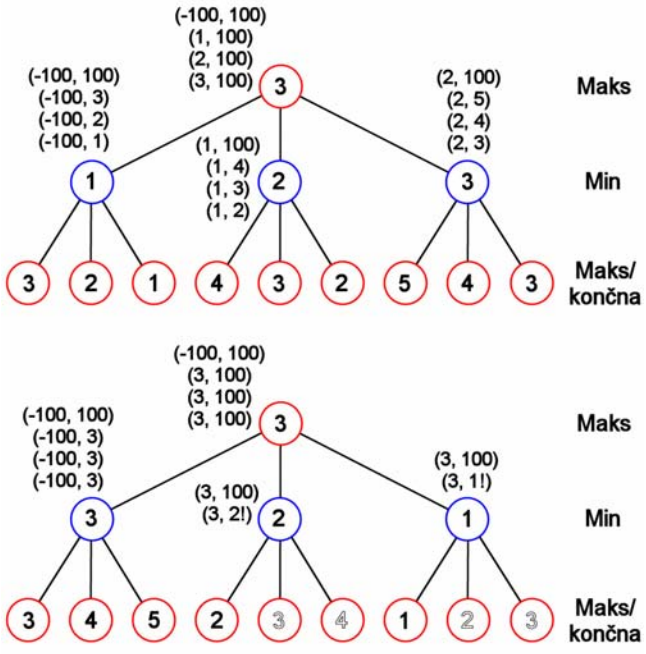
2.2. Transpozicijska tabela in particijsko iskanje

Pogosto se zgodi, da se v drevesu večkrat pojavi isto stanje igre – npr. če dve bolj ali manj neodvisni potezi naredimo v različnem vrstnem redu, obakrat pridemo do istega stanja. Če si prvič to stanje zapomnimo, nam naslednjič ni treba preiskovati drevesa pod njim, ampak zgolj preberemo njegovo vrednost iz tabele. [3] Ta tabela se imenuje transpozicijska (*transposition table*) in je navadno implementirana kot zgoščena tabela.

Mnogo stanj, ki so shranjena v transpozicijski tabeli, si je zelo podobnih, zato bi jih bilo zaželeno združiti. To izkorišča particijsko iskanje (*partition search*) [4], ki v tabelo shranjuje množice vozlišč, za katera izračuna, da jim pripada enaka ocena. Uporablja ga GIB, trenutno najbrž najboljši program za igranje bridža.

2.3. Druge izboljšave

Iskanje se močno pospeši, če v vsakem vozlišču najprej preiščemo najboljšega naslednika. Meja iskanja se namreč v tem primeru že po prvem nasledniku nastavi na končno vrednost za trenutno vozlišče in pri neprvih vozliščih hitro pride do rezov. Slika 3 kaže koristnost razvrščanja – vozlišča v zgornjem drevesu so razvrščena slabo (rezov sploh ni), v spodnjem pa dobro (rezi so povsod, kjer je možno). Razvrščamo lahko z iterativnim poglabljanjem, kjer se rezultati vsake iteracije uporabijo za razvrščanje naslednje. Že omenjen način je transpozicijska tabela, ki se lahko uporabi za razvrščanje, če podatki v njej ne zadoščajo za uporabo vrednosti iz tabele. Ker včasih utegne biti premajhna, tako da se veliko položejev prepiše, se uporabljaja tudi ovržbena tabela (*refutation table*) [3], ki shranjuje le poteze, ki so v preteklosti povzročile reze. Podobno ubijalska hevristika (*killer heuristic*) [5] za vsako plast drevesa hrani nekaj najboljših potez, ki jih nato vedno preizkusimo najprej. Zgodovinska hevristika (*history heuristic*) [6] pa je njena posplošitev in za vse poteze hrani vrednosti, ki merijo njihovo kvaliteto in so podlaga za razvrščanje. Na voljo pa so seveda tudi metode, ki temeljijo na znanju o problemu, s katerim se ukvarjamo.



Slika 3: Razvrščanje potez pri algoritmu alfa-beta

Če iščemo z oknom manjšim od (α, β), lahko prihranimo nekaj časa, čeprav se utegne zgoditi, da moramo iskanje kdaj ponoviti z večjim oknom. Z mejama blizu pričakovane vrednosti išče aspiracijsko iskanje (*aspiration search*) [7], njegova skrajna različica pa je iskanje z najmanjšim oknom (*minimal windows search*) [3], ki je učinkovito, če razvrščamo poteze. Prvega naslednika vozlišča namreč preiščemo s polnim oknom, vse nadaljnje pa z oknom širine (n, n+1), kjer je n rezultat prvega, saj skušamo le pokazati, da so slabši od prvega.

Prilagajamo lahko tudi globino iskanja – zanimivejša vozlišča preiščemo globlje. Vendar tu kakih splošnih metod skoraj ni.

3. OBRAVNAVANJE NEPOPOLNE INFORMACIJE

Nepopolno informacijo v igrah si lahko predstavljamo kot npr. nabor različnih možnih razporeditev kart med igralce (recimo temu množica svetov), igralci pa ne vedo, katera je prava (v katerem svetu so). [8] Drevo igre z nepopolno informacijo je podobno drevesu igre s popolno informacijo, le da imajo listi po več vrednosti (za vsak svet svojo).

3.1. Vzorčenje Monte Carlo

Vzorčenje Monte Carlo (*Monte Carlo sampling*) [8] naključno izbere nekaj svetov in preišče drevo igre zanje. Načeloma bi lahko preiskali vse svetove, a to je praviloma nepraktično, ker jih je preveč. Če najdemo najboljšo rešitev v dovolj svetovih, je verjetno, da je najpogostejša tudi zares najboljša. Funkcija za vrednotenje potez je takšna:

$$f(\text{poteza}_i) = \sum_{j=1}^{n} P(s_j) \times \text{vrednost}_{ij}$$

$P(s_j)$ verjetnost, da je izmed n svetov pravi j-ti, vrednost_{ij} pa za i-to potezo in j-ti svet izračunamo z algoritmom minimaks ali kako njegovo izvedenko. $P(s_j)$ je navadno enaka 1. Če ni, je to selektivno vzorčenje (*selective sampling*) [7], ki da enako dobre rezultate pri manjšem številu vzorcev (če seveda izberemo reprezentativne svetove), a ga je težje implementirati.

3.2. Težave vzorčenja Monte Carlo

Poleg tega, da vzorčenje Monte Carlo utegne dati napačne rezultate zaradi svoje statistične narave, ima tudi težave, ki bi nastopile celo, če bi preiskali vse svetove. [8]

Prva je, da se *maks* obnaša, kot da ima *min* popolno informacijo, čeprav je v resnici navadno nima in bi *maks* to lahko izkoristil. Druga je, da vzorčenje Monte Carlo ravna, kot da bi bi bilo odločanje v vsakem vozlišču odvisno le od drevesa pod njim, kar pa ni nujno res – če *min* pozna pravi svet in *maksovo* delovanje, lahko izbira tako, da se *maks* znajde pred odločitvami, kjer možnost, ki se z verjetnostnega vidika zdi najboljša, ni prava. A glede na to, da pravzaprav ne vemo, koliko *min* ve, ti dve težavi težko rešujemo (za drugo celo obstaja algoritem, a ni dovolj učinkovit za praktično rabo).

Tretja težava pa je prelaganje odločitev. Npr. izbiramo med potezama A in B. Z A zmagamo, če ima škisa prvi nasprotnik in bo naša naslednja poteza C ali pa če ima škisa drugi nasprotnik in bo naša naslednja poteza D. Z B pa zmagamo ne glede na to, kdo ima škisa, razen če ima vse srčeve karte prvi nasprotnik (kar je malo verjetno). Vzorčenje Monte Carlo bo kot pravilno potezo izbralo A, ker z njo lahko zmagamo v vsakem primeru. Vendar to velja ob predpostavki, da bomo do naslednje poteze vedeli, kdo ima škisa, kar pa se najbrž ne bo zgodilo, tako da bi bilo pametneje igrati B. To rešimo tako, da poiščemo čim večjo množico svetov, v kateri lahko zmagamo v vsakem primeru, nato pa igramo, kot da je pravi svet v tej množici. [4] To metodo uporablja že omenjani GIB, a v primerjavi z navadnim Monte Carlom ne prinaša nistvenih izboljšav.

4. TAROK

4.1. Opis igre

Moj program igra tarok za tri igralce [9]. Vsak dobi 16 kart, šest pa jih gre v talon. Nato igralci licitirajo, kakšno igro bodo igrali (koliko kart bodo založili – to pomeni, koliko jih bodo iz talona uvrstili v list, nato pa enako število iz lista dali med pobrane): kdor se odloči za najtežjo igro, zmaga na licitaciji, ostala dva pa sta njegova nasprotnika. Vsak krog igralci odvržejo po eno karto in tisti z najvišjo pobere vse tri. Na barvo mora vsakdo vreči isto barvo, če jo ima, sicer taroka (to so karte s številkami od ena do 21 in škis, ki šteje kot 22), če nima niti tega, pa karkoli. Višje karte poberejo nižje, taroki pa poberejo barve. Zmaga tisti, ki pobere več vredne karte (to so visoke barve, taroka ena in 21 ter škis).

4.1 Zasnova programa

Ponujata se dva načina, kako se lotiti računalniškega programa za igranje iger: z rabo človeškega znanja in z grobo računsko silo. Pri prvi možnosti bi moral v program prenesti čim več človeškega znanja o taroku, na podlagi katerega bi ta potem igral. Pri drugi možnosti pa bi programu podal samo pravila igre, ta pa bi potem preiskoval poteze, ki jih lahko naredi, in nasprotnikove odgovore nanje ter se na podlagi tega odločal, kako igrati.

Pretežna raba človeškega znanja ima slabosti: potrebni so dobri strokovnjaki, tak program utegne biti predvidljiv in neprilagodljiv, pa tudi mnoga znanja je težko prenesti v program. Glede na to, da sta pri igranju iger glavni prednosti računalnika pred človekom velika hitrost in popoln spomin, ju velja čim bolj izkoristiti. Če bi bil računalnik zmožen v dovolj kratkem času preiskati vse drevo igre, bi to že zadoščalo. Vendar žal ni tako – temu cilju se niti zelo približal nisem, tako da najbrž težava ni samo v tem, da nisem izdelal dovolj dobrega preiskovalnega algoritma, ampak je to za zdaj pretežak problem. Zato sem se sicer še vedno osredotočil na hitro preiskovanje možnosti, a sem ga obogatil z rabo človeškega znanja, kjer je bilo potrebno.

4.2. Algoritmi

Prva naloga programa je licitiranje, ki zahteva, da oceni moč kart, ki jih ima v listu – da ugotovi, v kakšni igri se z njimi da zmagati. Seveda bi se to dalo storiti tako, da bi se na podlagi števila tarokov, kraljev idr. izračunala ocena lista. Iz te ocene bi se nato določilo, kakšno igro je še moč dobiti, in program bi jo poizkusil izlicitirati. A to bi zahtevalo rabo človeškega znanja, pa tudi težavno bi bilo, ker je moč lista odvisna od mnogih dejavnikov. Zato sem se v skladu s svojo namero izogibati se rabi človeškega znanja raje zatekel k drugačni metodi: program moč lista oceni tako, da tvori več naključnih talonov, se za vsakega založi in simulira igro. Povprečje rezultatov simuliranih iger pove, kako dober je preizkušani list. Začne z najlažjo igro in napreduje proti težjim, dokler ne pride do take, za katero oceni, da v njej ne more zmagati; predzadnjo nato poizkusi izlicitirati. Ker simulacija zaradi časovnih omejitev ne more biti posebej temeljita in ker so zmožnosti nasprotnikov različne, se rezultat, ki se zahteva, da program oceni, da igro lahko dobi, prilagaja glede na uspehe pri dejanski igri.

Naslednja naloga je izbira kart iz talona in zalaganje. Pri izbiri kart iz talona ni smotrno upoštevati le kart samih, ampak tudi to, kako bodo dopolnile list in kako ugodno se bo dalo z dopolnjenim listom založiti. Zato program preizkusi vse skupine kart iz talona, se vsakič založi in simulira igro. Tisto skupina kart, kjer je rezultat simulacije najboljši, zares izbere. Pri zalaganju pa s človeškim znanjem v grobem določi možnosti (prav gotovo je npr. dobro ne imeti kart kake barve), nato pa jih temeljiteje preizkusi s simulacijo – taisto, ki je tudi podlaga za izbiro kart iz talona.

Za samo igranje program uporablja različico algoritma alfa-beta. Dopolnjen je s transpozicijsko tabelo, ki uporablja osnovno idejo particijskega iskanja: namesto posamičnih se vanjo vpisujejo množice vozlišč drevesa igre. Zaradi težavnosti implementacije pravega particijskega iskanja pa se množice enakovrednih vozlišč določajo hevristično (ne upošteva se, katere nizke karte so v listu, ampak le njihovo število; ne upošteva se, kateri taroki so v listu, ampak le njihovo število in vsota na 10 natančno). To v primerjavi z običajno transpozicijsko tabelo število razvitih vozlišč razpolovi. Uporablja tudi razvrščanje potez z zgodovinsko hevristiko in iskanje z najmanjšim oknom. Poleg tega pa prilagaja globino iskanja: na podlagi človeškega znanja

izloča nekatere nekatere veje drevesa. To dela le pri tretjih kartah vzetka, ko je o položaju znanega dovolj, ker je pri prvih ali drugih zelo težko določiti, katere niso zanimive. Ocenjevalna funkcija je taka:

f (stanje) = razlika × 5 + vsota_tarokov + tarok21 × 5 × × 10 + škis × 5 × 10

V enačbi razlika označuje razliko med vrednostjo pobranih kart programa in njegovega morebitnega soigralca ter nasprotnikov. Z vsota_tarokov je označena vsota programovih tarokov in preprečuje preveč pogumno odmetavanje tarokov, saj bi bilo slednje sicer dobro ocenjeno, ker se z visokimi taroki navadno pobere. Spremenljivki tarok21 in škis pa označujeta prisotnost teh dveh kart v programovem listu. Ker sta karti sami veliko vredni in ker se z njima navadno pobere, se je brez tega redno dogajalo, da ju je program uporabil ob prvi priložnosti, namesto da bi ju bil prihranil za kako pomembnejšo.

Nepopolno informacijo program obvladuje z vzorčenjem Monte Carlo, ki kljub opisanim težavam deluje precej zadovoljivo, obenem pa je preprosta in elegantna rešitev. Pri tvorbi vzorčnih razporeditev kart se upoštevajo vsi gotovi podatki, ki jih program ima o kartah drugih igralcev. Na začetku išče do globine devet, ko imajo igralci še po 10 kart, to poveča na 12, pri sedmih kartah pa na 15. Za vsako razporeditev kart se poleg iskanja do največje globine išče še do globine enega vzetka. Rezultati tega iskanja pri končni odločitvi veljajo 0,7-krat toliko kot rezultati iskanja do polne globine. To je koristno zato, ker se sicer postavi predpostavko o stanju igre, po kateri je zaporedje slaba, a neizogibna poteza – dobra poteza ocenjeno enako kot dobra poteza – slaba, a neizogibna poteza. Včasih zato najprej naredi slabo, a neizogibno potezo, za katero se zatem izkaže, da ni bila neizogibna. Plitvejše iskanje daje prednost takojšnjemu dobičku in take poteze prepreči.

4.3. Rezultati

Da se pokaže, kolikšni so prispevki posamičnih izboljšav algoritma alfa-beta in njihovih kombinacij, sem preštel, koliko vozlišč se razvije pri enem iskanju, in izmeril, koliko časa je potrebnega zanj. Nato sem še izračunal, koliko časa se porabi na vozlišče, da se vidi, koliko pribitka pri vsakem vozlišču povzroči katera izboljšava. Rezultati (na sliki 4) veljajo za prvo karto, igrano v igri, pri globini iskanja devet. Časi so bili izmerjeni na računalniku s procesorjem Athlon XP P1800+ in 512 MB pomnilnika.

Vse štiri izboljšave skupaj povzročijo, da se preišče 184-krat manj vozlišč, kot bi se jih z golim alfa-beta, za kar se porabi 86-krat manj časa (kar je posledica tega, da vse izboljšave podaljšajo čas, ki se porabi za eno vozlišče, za malo več kot dvakrat). Očitno je, da ima največji učinek transpozicijska tabela, celo če upoštevamo, da najbolj podaljša čas za eno vozlišče. Ostale tri izboljšave so si precej podobne, čeravno zgodovinska hevristika je nekoliko slabša od ostalih dveh.

Človeški igralci program ocenjujejo kot spodobnega, čeprav ne ravno vrhunskega nasprotnika. Igra dovolj dobro in nepredvidljivo, da je proti njemu zanimivo igrati. Vseeno dela včasih napake, pa tudi nekoliko brazciljno utegne igrati, ker preiskovanje ne seže dovolj globoko, da bi lahko naredil kak dolgoročen načrt. Opažene so bile tudi poteze, ki bi jih pri človeku označili za zvite. Npr. igralca pred programom sta vrgla nizki barvni karti, program pa je pobral s kavalom, čeprav je imel kralja. Naslednji vzetek je potem pobral s tem kraljem.

Algoritem	Vozlišča	Čas (s)	Čas/vozl. (µs)
vse izboljšave	8.667	0,248	28,6
brez transpozicijske tabele	25.349	0,527	20,8
brez zgodovinske hevr.	13.467	0,350	26,0
brez najmanjšega okna	15.827	0,433	27,4
brez izločanja vej	15.952	0,435	27,3
transpozicijska tabela	95.575	2,451	25,6
zgodovinska hevristika	324.633	5,463	16,8
najmanjše okno	320.792	4,438	13,8
izločanje nekaterih vej	258.060	4,891	18,9
nič izboljšav	1.598.924	21,266	13,3

Slika 4: Učinkovitost izboljšav algoritma alfa-beta

5. SKLEP

Izdelani program zadovoljivo igra tarok, do vrhunske igre pa mu še precej manjka. Preiskovalni algoritem, ki je sicer dober, bi se verjetno dal še izboljšati, dvomim pa, da bi bilo to dovolj. Najbrž bi se bilo treba zateči k rabi človeškega znanja in vdelati nekaj dolgoročnih strategij, s katerimi bi si pomagal v začetnem delu igre.

LITERATURA

[1] P. Y. Chan, H. Y. Choi, Z. Chiao. Data Structures and Algorithms Class Notes: Game trees, Alpha-beta search. http://www.cs.mcgill.ca/~cs251/OldCourses/1997/topic11, 1997

[2] G. P. Ingargiola. UGAI Workshop Lectures: MiniMax with Alpha-Beta Cutoff. http://yoda.cis.temple.edu:8080/UGAIWWW/lectures96/search/minimax/alpha-beta.html, 1996

[3] T. A. Marsland. A Review of Game-Tree Pruning. Journal of Computer Chess Association 9(1), 1986

[4] M. L. Ginsberg. GIB: Imperfect Information in Computationally Challenging Game. Journal of Artificial Intelligence Research 14, 2001

[5] R. Feldmann. Game Tree Search on Massively Parallel Systems. Advances in Computer Chess 7, 1993

[6] J. Schaeffer. The History Heuristic and Alpha-Beta Search Enhancements in Practice. IEEE Transactions on Pattern Analysis and Machine Intelligence, 1989

[7] J. Schaeffer. The Games Computers (and People) Play. Advances in Computers 50, 2000

[8] Frank & D. Basin. A Theoretical and Empirical Investigation of Search in Imperfect Information Games. Theoretical Computer Science, 1999

[9] Spletna stran Tarok zveze Slovenije. http://users.volja.net/tarokzveza