Poskus indukcije človeku razumljivih pravil z atributnim učenjem pri šahovskih končnicah

Aleks Jakulin

30. avgust 2001

Kazalo

1 Šahovske končnice

Klasične metode umetne inteligence za igranje iger temeljijo na preiskovanju. Postopki iskanja temeljijo na tem, da definiramo hevristične ocene na podlagi števila figur in ugodnosti njihovega položaja na šahovnici. Potem pa izberemo tiste poteze, ki nas v najslabšem primeru pripeljejo do najboljše ocenjenega položaja. Tiste poteze, ki nas pripeljejo do tega, da izgubimo figuro, bodo ocenjene slabše kot pa poteze, pri katerih bo figuro izgubil nasprotnik. Seveda moramo gledati nekaj potez v prihodnost, da bi uspešno tekmovali s človekom.

Čeprav tak način igranja iger omogoča računalniku zmago v primerjavi s katerimkoli človekom, moramo priznati, da so omenjene metode tako specifične, da bi težko govorili o podobnem tipu inteligence, kot jo ima človek. Človek se namreč ob igri uči, si ustvarja nove koncepte in izkušnje, na podlagi katerih lahko brez velikega števila korakov računanja dosega podobne rezultate.

Strojno učenje poskuša na podlagi danih primerov priti do modelov, ki opisujejo lastnosti primerov. Ti modeli lahko služijo za predvidevanje lastnosti primerov, tudi če teh ne poznamo. Če so modeli dobri, tudi dokumentirajo naravo problema in s tem človeku omogočijo razumevanje.

Atributno učenje je podvrsta strojnega učenja, kjer za vsak primer navedemo vse njegove lastnosti, ki jih poznamo. Lastnosti delimo na dve vrsti: razred je lastnost, ki bi jo radi predvideli, atributi pa so lastnosti, ki so dane.

Šahovske končnice so primerne za obravnavo s strojnim učenjem zaradi vrste razlogov. So najenostavnejši zaokroženi podproblemi v šahu. Dokler metode strojnega učenja ne bodo sposobne obravnavati takih, tudi bolj zapletenih ne bodo. Končnice lahko dokaj enostavno analiziramo z retrogradno analizo. To pomeni, da za vsak končni položaj v igri (mat, remi) analiziramo možne poteke igre v nasprotni smeri. Kot rezultat te analize, pripravimo bazo, ki za vsak položaj v igri pove, kako kvaliteten je, oziroma, koliko potez je najmanj potrebnih za zmago privilegirane strani v najslabšem primeru. Končno, klasični postopki igranja iger niso najbolj primerni za obravnavo končnic, saj usmerjevalne hevristike niso enostavne ali očitne, če pa teh hevristik ne določimo, je potrebna globina iskanja precej velika. V praksi potem programi za igranje šaha uporabljajo obširne baze podatkov, dobljene z retrogradno analizo. Na podlagi komentarjev k igri programa Deep Blue lahko sklepamo, da so šahovski programi z uporabo takih podatkovnih baz boljši od človeka, ter da je igra v končnici ena izmed močnejših točk programa.

1.1 O končnici KRK

S kratico KRK bomo označevali končnico, pri kateri poskušata beli kralj in trdnjava matirati samega črnega kralja. Cilj za uspešnost strategije je, da vedno pride do mata in to v manj kot 50 potezah. Omejili se bomo na atributno učenje in z njim poskušali priti do uporabne strategije.

1.2 Struktura baze

Čeprav je vseh možnih dovoljenih položajev v končnici KRK 249984, obstajajo med njimi simetrije, ki nam število položajev na enostaven način zmanjšajo na 28056 položajev. To storimo tako, da belega kralja z zrcaljenjem preslikamo v spodnji levi kvadrant, pri čemer poleg belega kralja zrcalimo tudi ostali dve figuri. Nato ga na isti način preslikamo še pod diagonalo od A1 do H8. Če je beli kralj na diagonali in črni kralj nad njo, preslikamo črnega kralja in trdnjavo pod diagonalo. Končno, če sta oba kralja na diagonali, trdnjava pa nad njo, preslikamo trdnjavo pod diagonalo.

Kot smo že prej omenili, za vsak položaj povemo, koliko potez bi beli v najslabšem primeru potreboval za mat, če je za dan položaj na potezi črni. Oziroma, po najmanj koliko potezah je lahko črni matiran. Položaji, kjer lahko črni z eno potezo pride do remija, so posebej označene z -1. Položaji, kjer je črni že matiran, so označene z 0. V najslabšem primeru potrebuje beli 16 potez do mata.

Tako bazo uporabljamo pri igri tako, da beli za vse možne poteze preuči oznako ciljnega položaja in izbere potezo, ki ga pripelje na položaj z najnižjo oznako ali do mata. Črni pa za optimalno obrambo izbere tisto potezo, iz katere beli potrebuje največ potez do zmage. Oznaka poteze v svojem bistvu označuje *kvaliteto* položaja iz vidika belega. Beli poskuša izbrati potezo, ki ga pripelje do najkvalitetnejšega položaja. Ne obravnavamo le kvalitete položajev, temveč tudi kvalitete potez. Kvaliteta poteze je enaka kot kvaliteta ciljnega položaja.

2 Strojno učenje

Uporabo strojnega učenja v primeru šahovskih končnic lahko izrazimo kot poskus zmanjševanja obsega baze podatkov s tem, da kvaliteto vsakega položaja ali poteze opišemo s *strategijo*. Strategija je lahko odločitveno drevo, program v programskem jeziku, ali pa skupek pravil. Strategija nam služi pri izbiri poteze v danem položaju, mora pa delovati za vsak dovoljen položaj, kar z drugimi besedami pomeni, da mora biti strategija polna ali kompletna.

V primeru, da ne izberemo najkvalitetnejše poteze, se lahko vprašamo, koliko manj kvalitetna je poteza, ki jo izbere strategija, v primerjavi z najkvalitetnejšo. Ker kvaliteto opisujemo s številom potez do mata, je *suboptimalnost* razlika med kvaliteto izbrane in kvaliteto najboljše poteze. Suboptimalnost je nasprotje učinkovitosti, pri čemer točno vemo, koliko so poteze, ki jih svetuje neko pravilo, v povprečju manj učinkovite od optimalnih. Lahko govorimo tudi o suboptimalnosti baze podatkov ali strategije, ki jo definiramo kot vsoto povprečnih suboptimalnosti izbranih potez v vseh položajih.

V primeru, da so naše poteze suboptimalne, se lahko zgodi, da ne zmagamo, saj lahko zaidemo v cikle, kjer v poteku igre večkrat pridemo v isti položaj. Zato je važen koncept *pravilnosti*. Pravilen potek igre nam zagotavlja, da bo beli iz poljubnega položaja vedno zmagal v manj kot 50 potezah. Moramo pa pravilnost strategije preveriti za celotno strategijo, saj na podlagi posameznega pravila še ne moremo sklepati o pravilnosti celote. Suboptimalnost je smiselno

obravnavati le, če imamo zagotovoljeno pravilnost. Med več pravilnimi strategijami pa bi načelno izbrali tisto z najmanjšo suboptimalnostjo.

2.1 Učenje ocen položajev

V večini prejšnjih poskusov uporabe strojnega učenja za končnico KRK je bil cilj učenja model, ki bi za dan položaj ocenil njegovo kvaliteto. Čeprav je to lahko regresijski problem, so bili poskusi Quinlana [4] in ostalih usmerjeni v ugotavljanje, ali je dan položaj ocenjen z manj ali enako n ali ne. Ocena manj ali enako n ustreza konceptu, da je igra izgubljena za črnega v n potezah. Take modele uporabljamo podobno, kot da bi uporabljali celotno bazo ocen položajev.

Quinlan je z uvedbo pomožnih konceptov in postopkom ID3. Za položaje z oceno manjšo ali enako 2 je bilo uporabljenih 23 atributov, dobljeno razvrščevalno (klasifikacijsko) drevo je imelo 83 vozlišč, za položaje z oceno manjšo ali enako 3 pa 39, pri čemer je imelo razvrščevalno drevo 177 vozlišč.

Bain v svojem doktoratu navaja pravilne modele za identifikacijo položajev z ocenami 0 in 1, dobljene z relacijskim učenjem, ne da bi človek definiral pomožne koncepte. Po interpretaciji naučenih vmesnih konceptov se ti nanašajo npr. na varnost ali ogroženost trdnjave v primeru, ko je črni kralj v šahu. Pri učenju modela za n so pozitivni primeri tisti, ki so ocenjeni z n, negativni pa tisti z oceno večjo od n. Samo učenje poteka tako, da od učnih primerov pri učenju izločamo primere, ki jih pokrijejo modeli za manjše n. Bain je sicer izdelal tudi modele do vključno $n = 5$, vendar kompleksnost pravil hitro narašča.

Quinlan je s svojim sistemom FOIL za relacijsko učenje [5] dobil pravilna pravila za položaje z ocenami 0, 2, 15 in 16. Pri položajih za ostale ocene so izjeme (pravilni primeri, ki pa jih pravila ne pokrijejo) navedene eksplicitno. Za oceno 11 je ena napaka, sicer pa je vseh pravil, vključno z izjemami, nekaj manj kot 2500. Nasploh se FOIL po podatkih sodeč obnaša bolje.

2.2 Učenje ocen potez

Človek ponavadi ne opisuje le položajev, temveč tudi lastnosti potez. Poleg tega, da imamo na ta način večje število atributov, ima ta pristop tudi druge prednosti. Strategija je bolje razvidna iz lastnosti potez, ki izražajo naše relativne želje, kot pa iz štetja, da bi prišli do numeričnih meril kvalitete.

Primer strategije, kot jo je definiral človek, je tista iz [2]. Z besedami bi jo opisali takole:

1. Poizkusi matirati nasprotnikovega kralja v dveh potezah. `(Mate)`
2. Če to ni mogoče, poizkusi bolj omejiti območje gibanja nasprotnikovega kralja, ki je omejeno s trdnjavo. `(Squeeze)`
3. Če tudi to ni mogoče, poizkusi približati kralja k nasprotnikovem kralju in s tem podpreti trdnjavo pri stiskanju nasprotnika. `(Approach)`
4. Če nobenega od zgornjih nasvetov ni mogoče uporabiti, poizkusi vsaj ohraniti dosežke v smislu točk 2 in 3. `(KeepRoom)`

5. Če noben od zgornjih ciljev ni dosegljiv, poskusi doseči pozicijo, v kateri trdnjava deli oba kralja, ali vertikalno ali horizontalno. `(Divide)`

Pri vseh potezah po vseh nasvetih mora biti nujno izpolnjen pogoj, da nasprotnikov kralj ne pride v pat ter da trdnjava ne postane izpostavljena (kar bi pomenilo, da je trdnjava po potezi belega ob črnem kralju, ne da bi jo ščitil beli kralj.)

Formalno lahko zgornje nasvete izrazimo z jezikom nasvetov (advice language). S tem namenom definiramo seznam predikatov, ki opisujejo ciljne pogoje in se nanašajo na položaje:

mate	črni kralj v matu.
stalemate	črni kralj v patu.
rooklost	črni kralj lahko vzame trdnjavo.
rookexposed	črni kralj lahko pride do trdnjave preden pride do nje beli kralj.
rookdivides	trdnjava horizontalno ali vertikalno loči kralja med seboj.
lpatt	vzorec v obliki črke L, kjer je v kolenu črni kralj, 3 polja daleč od njega je trdnjava, 2 polji daleč pa je beli kralj.
roomgt2	prostor, ki ga ima na voljo črni kralj, je večji od 2 polj.
tkingedge	črni kralj je na robu.
okingedge	beli kralj je na robu.

Poleg tega pa imamo tudi predikate, ki opisujejo potezo:

depth	globina poteze v iskalnem drevesu.
legal	kakršnakoli dovoljena poteza.
checkmove	kakršnakoli poteza, ki povzroči šah.
rookmove	poteza trdnjave.
nomove	neuspeh za vsako potezo.
kingdiagfirst	poteza kralja, najprej poskusi z diagonalnimi potezami.
okapproachedcsquare	beli kralj se je približal kritičnemu polju, pri čemer je kritično polje tisto, kamor se mora črni kralj premakniti, da bi se čimbolj oddaljil od trdnjave.
okorndle	beli kralj in trdnjava po potezi nista bolj oddaljena (v kraljevih potezah), kot sta bila pred potezo.
roomsmaller	prostor, ki ga ima črni kralj na razpolago, je po potezi manjši.

Nekateri od teh predikatov niso najbolj razumljivi brez razlage jezika nasvetov AL0, ki ga najdemo v [2] in ga tukaj ne bomo definirali. V osnovi pa AL0 deluje na principu ocenjevanja in priporočanja potez, pri čemer so njegovi nasveti urejeni po pomembnosti.

Pravilnost strategije dokažemo tako, da se po uporabi vsakega pravila prostor, ki ga ima na voljo črni, stalno zmanjšuje. Groba zgornja meja za zmago belega je 39 potez.

Ob delovanju strategije lahko opazimo, da je enostavno pravilo **Squeeze** bistveno za uspešnost. Ostala pravila le preprečujejo, da bi si bistveno pokvarili izhodišče, med tem pa skoraj naključno in izrazito suboptimalno iščejo položaj, iz katerega se bo dalo ponovno uporabiti **Squeeze** ali **Mate**.

Vprašanje je, kako formalno predstaviti ocene potez, da bi jih lahko obravnavali kot problem za atributno učenje. Recimo, da lahko vsako možno pravilo opišemo kot konjunkcijo vrednosti atributov. Atribute lahko razdelimo v tri skupine: lastnosti začetnega položaja, lastnosti poteze ter lastnosti končnega položaja. Razred pa označuje, ali je pravilo dobro ali ne: dobra pravila nam povedo, kakšne morajo biti lastnosti poteze in končnega položaja na podlagi znanih lastnosti začetnega položaja. Tako predstavljeni množici dobrih pravil pravimo osnovna pravila.

Cilj učenja je model, ki dobra pravila loči od slabih, na podlagi dobrih pravil pa najdemo dobre poteze. Pri igri model uporabimo tako, da poskusimo povleči tiste poteze, ki jo priporočajo dobra pravila. Torej, za trenutni položaj izračunamo vrednosti atributov, potem za vsako možno potezo izračunamo še atribute poteze ter atribute končnega položaja. Na tak način imamo določene vse atribute in na podlagi pravil izvemo, ali je poteza dobra ali slaba.

3 Pristop in rezultati

Odločili smo se za učenje tega, kako kvalitetne so posamezne poteze, namesto učenja kvalitete položajev, saj strojno učenje kvalitete potez prej sploh še ni bilo preizkušeno. Samo delo lahko razdelimo na več nalog, od priprave atributov, ocenjevanja kvalitete osnovnih pravil, ter končno, učenja enostavnejših pravil na podlagi ocenjenih osnovnih pravil.

3.1 Priprava atributov

Strategija v AL0 ni neposredno primerna za atributno učenje, saj vključuje elemente preiskovanja v globino, generiranja možnih potez, ki jih z zgoraj omenjenim atributnim pristopom ne moremo doseči. Zato moramo dodati nekaj novih atributov.

Atributa `nomove` in `depth` nista več uporabna. Atributi o potezah `rookmove`, `newroomsmaller`, `okapproachedcsquare`, `okorndle` in `checkmove` ostanejo nespremenjeni, kot tudi atributi položajev `mate`, `stalemate`, `rooklost`, `rookexposed`, `rookdivides`, `lpatt`, `tkingedge`, `okingedge` ter `roomgt2`.

Nekaterih AL0 pravil v [2] ravno tako ne moremo pretvoriti v navadna atributna pravila. To lahko rešimo tako, da dodamo nekaj novih atributov. Predelati moramo atribut `kingdiagfirst`, ki ga nadomestimo z dvema: `kingorth` in `kingdiag`. To nam omogoči, da vsak prologov predikat, ki bi uporabljal `kingdiagfirst`, nadomestimo z dvema atributnima praviloma.

Mat v dveh potezah in deljenje kraljev v več potezah pa je zapletenejši problem, saj temelji na preiskovanju v globino. Tega v kontekstu pravil ne moremo doseči, lahko pa pomagamo z dodatnimi atributi, ki bi omogočili vmesne korake brez iskanja. Za mat v dveh potezah smo uporabili kar originalno bazo kvalitete

potez, ki se nanaša na položaje, lahko pa bi tudi uporabili sintetične atribute, kot jih dokumentira, npr. Bain v [1].

Za deljenje pa po [3] deluje postopek, kjer v prvih dveh potezah premaknemo trdnjavo tako, da bo njena horizontalna ali vertikalna razdalja do belega kralja enaka 1, v naslednji potezi pa lahko premaknemo trdnjavo tako, da je izpolnjen pogoj `rookdivides`. Očitno potrebujemo nov atribut `rookhome`, ki velja, ko je horizontalna ali vertikalna razdalja med belim kraljem in trdnjavo enaka 1. Za dosego cilja `rookhome` potrebujemo dve potezi le v primerih, ko so vse tri figure na diagonali, pri čemer je beli kralj v kotu, eno polje daleč je črni kralj, zraven njega pa je bela trdnjava (npr. WK: a1, BK: c3, WR: d4). V takem primeru se mora bela trdnjava najprej odmakniti v tak položaj, da bo po potezi od črnega kralja po manhattanski metriki oddaljena vsaj dva polja. Za take položaje definiramo atribut `rooksafe`. Z omenjenimi novimi atributi, lahko strategijo predstavimo z urejenim seznamom pravil, ki so sestavljena le iz atributov.

Dobljene strategije smo vrednotili tako, da smo jih uporabili v partijah iz naključnih položajev proti minmax optimalnemu črnemu nasprotniku.

Samo učenje smo izvajali na celotni množici vseh možnih potez, pri čemer smo za vsako vedeli njeno suboptimalnost. Današnji računalniki so dovolj hitri, da izkoriščanje simetrije ne prinaša pomembnih koristi.

3.2 Pravila

3.2.1 Dobra in slaba pravila

Najprej smo poskusili, kako se obnese enostavna delitev vseh osnovnih pravil na dobra in slaba. Za vsako osnovno pravilo lahko namreč ocenimo, koliko povprečno je suboptimalnost vseh potez, ki jih osnovno pravilo pokriva. Če je suboptimalnost manjša od določene stopnje, na primer 2 ali 1, je pravilo dobro, sicer ni.

Zaradi omejenega števila atributov si ne moremo privoščiti le optimalnih pravil, saj le ta pokrijejo le delček odstotka vseh položajev. Marsikateri položaj ni imel nobene poteze, ki bi jo priporočalo kako dobro pravilo, kar smo reševali z naključnimi a dovoljenimi potezami.

Igra programa s takimi pravili niha med večinoma dobrimi potezami, ki pa jih prekinejo slabe, te pa z eno potezo tako pokvarijo položaj, da potrebuje več potez za vrnitev na boljši položaj, če se vmes kaj podobnega ne zgodi še enkrat.

Kljub slabi igri je postopek služil kot osnova za poenostavljanje dobljenih osnovnih pravil v človeku razumljiva z uporabo programa CN2, kar bomo opisali v naslednjih razdelkih.

3.2.2 Preprečevanje vračanja

Včasih program zapade v ciklično igro. To smo rešili s preprečevanjem vračanja v že obiskane položaje, vendar to ni rešilo problema slabe igre.

3.2.3 Zagotavljanje polnosti

Lahko se zgodi, da v kakem položaju nimamo nobenega dobrega pravila. Zato za take položaje označimo kot dobro najboljše pravilo, ki se v tem položaju pojavlja. Čeprav smo s tem bistveno zmanjšali količino naključne igre, se kvaliteta igranja ni bistveno povečala. Osnovni problemi nihanja med dobrimi in slabimi potezami ostajajo.

3.2.4 Pomembnost pravil

Pravila niso le dobra ali slaba, marveč so urejena po pomembnosti. Od relativne pomembnosti pravil je tudi odvisna pravilnost strategije kot celote. Začetni poskusi so temeljili na neurejenih pravilih. Zato smo poizkusili osnovna pravila urediti. Uredili smo jih kar po povprečni suboptimalnosti. Uporaba urejenih pravil temelji na tem, da poskušamo povleči tisto potezo, ki jo svetuje najboljše pravilo.

Vendar pa tak pristop nosi v sebi nevarnost: pravila smo vrednotili po njihovi povprečni suboptimalnosti na vseh možnih potezah, ki jih pravila pokrivajo. V primeru, da so pravila urejena, v marsikaterem položaju izberemo le poteze, ki jih priporočajo pravila z najvišjo prioriteto. V takih položajih ostala pravila ne bodo nikdar uporabljena, zato tudi ni prav, da to suboptimalnost upoštevamo. Problem rešimo tako, da po izbiri najboljšega pravila ignoriramo vse tiste položaje, ki jih to pravilo pokrije.

Dobljena strategija pokriva 929 osnovnih pravil od 3500 možnih. Igra je smiselna, usmerjena, vendar pa program občasno naredi napake, iz katerih se veliko časa pobira. Primer igre iz položaja beli Kb7 Rd7 in črni Kc5. Igro začne beli in poteka takole:

```
1. Rd7 d8 (deset potez do mata)
 2. Kc5 b4
3. Kb7 c6
 4. Kb4 c4
5. Rd8 d5
 6. Kc4 b3
7. Kc6 b5
 8. Kb3 c3
9. Rd5 d6
 10. BKc3 b2
11. Kb5 c4
 12. Kb2 c2
13. Rd6 d3 (sledi serija slabih potez, s = -1)
 13. BKc2 b2
14. Rd3 d7 (s = -1)
 15. Kb2 c2
16. Kc4 c5 (s = -3)
 17. Kc2 b3
18. Rd7 d4 (s = -2)
 19. Kb3 c3
```

```
20. Rd4 d8
 21. Kc3 b3
22. Rd8 c8 (s = -2)
 23. Kb3 c2
24. Rc8 b8 (s = -2)
 25. Kc2 d1
26. Rb8 b2 (konec slabih potez)
 27. Kd1 c1
28. Rb2 f2
 29. Kc1 d1
30. Kc5 d4
 31. Kd1 e1
32. Kd4 e3
 33. Ke1 d1
34. Rf2 a2 (s = -1)
 35. Kd1 c1
36. Ke3 d3
 37. Kc1 b1
38. Ra2 c2
 39. Kb1 a1
40. Kd3 c3
 41. Ka1 b1
42. Kc3 b3 (optimalna igra zadnjih dveh potez)
 43. Kb1 a1
44. Rc2 c1
```

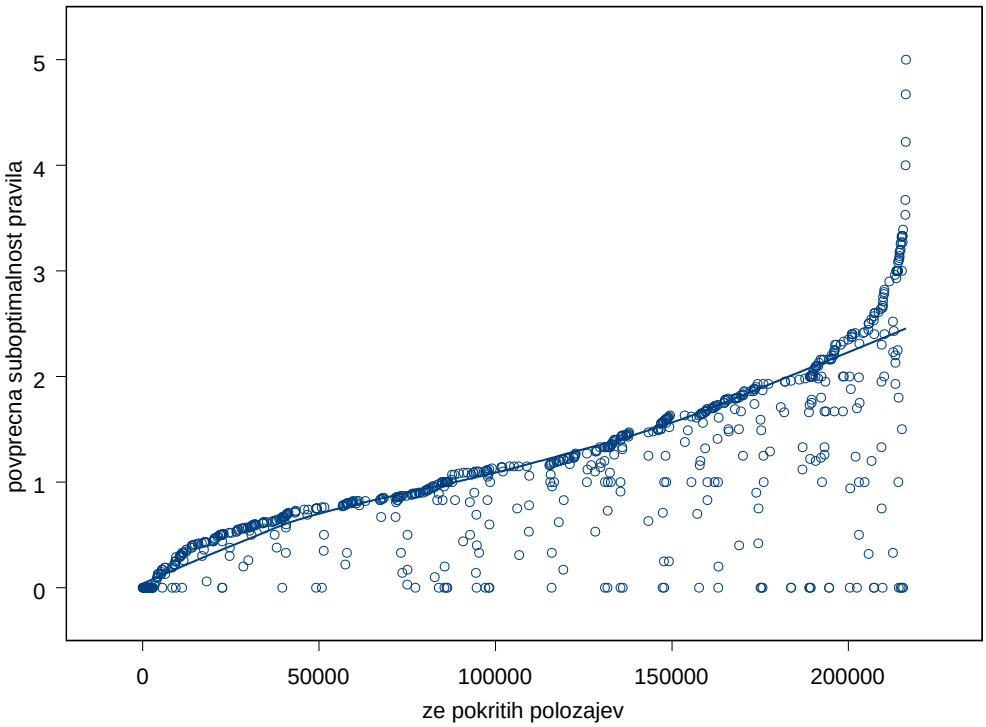
Primer uspešne igre iz beli Kg2, Rc8, črni Kb2, na potezi je beli, pa je tak:

```
1. Kg2 f3
 2. Kb2 b3
3. Kf3 e4
 4. Kb3 b4
5. Ke4 d5
 6. Kb4 b3
7. Rc8 c4
 8. Kb3 a3
9. Kd5 c5
 10. Ka3 b3
11. Kc5 b5
 12. Kb3 a2
13. Kb5 b4
 14. Ka2 b2
15. Kb4 a4 (zapravljena poteza)
 16. Kb2 a2
17. Rc4 b4
 18. Ka2 a1
19. Ka4 b3 (konec igre po pravilih, dve potezi do mata)
 20. Ka1 b1
```

```
21. Rb4 c4
 22. Kb1 a1
23. Rc4 c1
```

Seveda pa obstajajo tudi partije, ko se taka strategija ne obnese dobro. Pri preverjanju mi sicer ni uspelo najti partije, kjer beli ne bi prišel do mata, vendar pa včasih potrebuje precej več kot 50 potez. Zato ta strategija ni pravilna.

Sicer pa je glavna pomanjkljivost dobljene strategije ta, da se ni uspešno naučila pravil stiskanja ter da velikokrat trdnjave ne ohranja med obema kraljema. Pogosto med stiskanjem "pozabi," da je beli kralj med trdnjavo in črnim kraljem.



Slika 1: Odvisnost kvalitete od vrstnega reda pravil pri osnovni strategiji. Iz grafa je na primer razvidno, da lahko 90000 položajev pokrijemo s pravili, ki imajo povprečno suboptimalnost vedno manjšo od 1. Če ne bi bilo odvisnosti med pravili, bi ta graf bil popolnoma monoton. Ker pa nekatera pravila pokrijejo položaje, v katerih so druga pravila slabša, postanejo ta pravila boljša v nadaljevanju izbire. Vidimo tudi, da je za popolno pokritje vseh položajev potrebnih nekaj dokaj slabih pravil.

3.2.5 Kratkovidnost suboptimalnosti

Omenjeni postopek nam da povprečno suboptimalnost poteze 1.2 - pri računanju smo pesimistični: če najboljše pravilo svetuje več potez, izberemo najslabšo. To je bistveno boljše od naključne poteze z suboptimalnostjo 2.4. Kljub temu pa moramo upoštevati, da se s tako suboptimalnostjo v vsakem položaju v povprečju *oddaljimo* od mata! Kako naj to razumemo?

Suboptimalnost lahko razumemo na dva načina: "dobra" suboptimalnosti se nanaša na daljšo a zanesljivo pot do cilja, "slaba" pa na poti, po kateri pa ne moremo naprej in se moramo vrniti nazaj. Dobra suboptimalnost pomeni na primer, da s trdnjavo zmanjšamo prostor kralja, vendar ne za maksimalen obseg. Slaba suboptimalnost pomeni, da nesmiselno premaknemo kralja, in ga moramo potem premakniti nazaj na isto mesto. Pri našem dosedanjem delu smo predpostavljali, da vedno gre za dobro suboptimalnost, čeprav se pojavlja tudi slaba.

Dobro suboptimalnost lahko seštevamo in povprečimo, tega pa ne smemo početi s slabo suboptimalnostjo. Povrh tega pa je slaba suboptimalnost dejansko večja, kot se izračuna: če je povprečna slaba suboptimalnost strategije n, ne bo za igro v položaju, kjer za optimalno igro zadošča m potez, potrebovala le nm potez, ampak lahko bistveno več. Namreč, po slabi potezi smo pravzaprav na začetku, ali pa celo na slabšem, kot smo bili na začetku.

Če bi obstajala le slaba suboptimalnost in če bi bila vsaka poteza enako suboptimalna, potem ne bi z našo 1.2-suboptimalno strategijo nikoli prišli do mata. A vendar ponavadi pridemo do mata. Če ne zaradi drugega, zato ker suboptimalnosti ne moremo nabrati neskončno: najslabši možni položaj je 16 potez do cilja in kakršnakoli poteza iz takega položaja nas privede v kvečjemu enako slab položaj.

Pri eksperimentih s strategijo se izkaže, da iz položajev, ki imajo več kot 12 potez do mata ni težko najti dobre poteze, ravno tako pri položajih, ki imajo 5 potez do mata ni težko do mata priti. Vendar pa v vmesnem območju strategija deluje pomanjkljivo. To lahko razumemo tako, da je na tem območju več slabe suboptimalnosti, ki je ne vrednotimo pravilno.

Opravili smo nekaj poizkusov, kjer smo pri računanju povprečja namesto vsote suboptimalnosti uporabili vsoto eksponentov suboptimalnosti. S tem smo zelo slabe poteze kaznovali veliko bolj kot srednje slabe. A rezultati so bili kvečjemu slabši. Mogoče zato, ker premalo nagradimo dobre poteze v primerjavi z neuporabnimi.

3.3 Človeku razumljive strategije

Seznam osnovnih pravil človeku ni razumljiv zaradi svojega obsega, kompleksnosti in neurejenosti. Prvi dve osnovni pravili, pri čemer prvo pokriva 6660 položajev, drugo pa 960, izgledata takole:

if not r-exposed & r-divide & room-gt2 & not l-patt & ok-edge & tk-edge & not rookhome **then try:**
move: room-small & not csquare & okorndle & not check & rookmove & not k-diag & not k-orth

state: r-exposed & r-divide & room-gt2 & not l-patt & ok-edge & tk-edge & not rhome

if not r-exposed & not r-divide & room-gt2 & not l-patt & not ok-edge & not tk-edge & not rookhome **then try:**
move: not room-small & not csquare & okorndle & not check & not rookmove & k-diag & not k-orth
state: r-exposed & r-divide & room-gt2 & not l-patt & not ok-edge & not tk-edge & rhome

Lahko pa jih združimo v enostavnejše sklope s postopki strojnega učenja. To smo tudi poizkusili z algoritmom CN2 za enostavni primer neurejenih pravil. Poskušamo sestaviti enostavnejša pravila, ki pravilno pokrivajo prostor osnovnih pravil. Torej, če obstajata dve dobri pravili, ki se ujemata v vsem, razen v tem, ali je naš kralj na robu ali ne, lahko položaj našega kralja izpustimo iz končnega pravila, ki ostane ekvivalentno, a enostavnejše. Na ta način smo dobili relativno obsežne množice več deset pravil. Primeri takih pravil so navedeni spodaj (ime atributa se začne na apriori, če se atribut nanaša na stanje položaja, move na potezo, posterior pa na položaj po potezi). Ta pravila predvsem pomenijo, česa ne smemo početi, v oglatih oklepajih pa je navedeno število osnovnih pravil, ki jih dano CN2 pravilo pokriva.

```
IF    apriori-our_king_edge = n
  AND apriori-their_king_edge = n
  AND posterior-rook-exposed = y
  AND posterior-our_king_edge = y
THEN  Class = n  [70 0]

IF    apriori-rook-exposed = n
  AND apriori-our_king_edge = n
  AND move-our-king-approached-csquare = y
  AND posterior-our_king_edge = y
THEN  Class = n  [84 0]

IF    apriori-rook-exposed = n
  AND move-our-king-approached-csquare = y
  AND move-rook-move = n
  AND posterior-our_king_edge = y
THEN  Class = n  [125 0]

IF    move-our-king-approached-csquare = y
  AND move-king-diagonal = y
  AND posterior-our_king_edge = y
THEN  Class = n  [66 0]
```

Zadnje pravilo pomeni to, da ne smemo vleči diagonalnih potez s kraljem proti kritičnemu polju na rob plošče. Predzadnje pravilo pa nam odsvetuje premikanje kralja na rob plošče takrat, ko trdnjava ni izpostavljena.

V primeru urejenih osnovnih pravil lahko sicer brez težav združujemo pravila, ki so med seboj neodvisna, torej, ki nikoli ne pokrivajo istega položaja. Da

pa bi združili odvisna pravila, bi morali omogočiti nekakšne meje tolerance, ki še dovoljuje združevanje osnovnih pravil z različnimi zaporednimi številkami. Seveda pa to lahko vpliva na pravilnost končne teorije, ki bi jo morali ob poenostavljanju sproti preverjati. Verjetno bolj smiseln pristop je izhajati iz enostavnih strategij in preverjati njihovo pravilnost, šele na koncu pa njihovo optimalnost.

4 Generiranje pravil z vzvratno analizo

Naučene strategije niso nujno pravilne. V literaturi, npr. [1] se obravnava napačno razvrščene primere le statistično, v smislu števila napačno razvrščenih položajev. Pozoren pa je treba biti na to, da lahko posledice napačno razvrščenih primerov pripeljejo do strategije, ki ne pripelje vedno do zmage belega, za razliko od tega, da bi bila strategija le suboptimalna. Taka strategija je praktično neuporabna.

Prva možnost je dokazovanje pravilnosti strategije, česar ni težko narediti, vendar pa se postavlja vprašanje, kaj storiti, če strategija ni pravilna.

Drugo možnost predstavljajo postopki učenja, katerih rezultat bi bile zagotovljeno pravilne strategije. To lahko dosežemo tako, da uporabljamo postopek učenja zaporedno od najlažjih (položajev zmage) navzven, pri čemer želimo pri vsakem koraku pripeljati kar največ položajev v že pokrite položaje na poti do zmage z uvedbo novih pravil. Posebej pa moramo obravnavati položaje, ki jih pravila pokrijejo, ne da bi jih usmerila v že pokrite položaje. Taki položaji so nezaželeni, čeprav ne pomenijo nujno neuspešne strategije. Moramo jih posebej pozorno opazovati, saj nas lahko pripeljejo do ciklov. Pravila so oštevilčena kar po vrstnem njihovega dodajanja.

Samo preiskovanje je lahko relativno hitro, saj nas zanimajo le nepokriti položaji, iz katerih so v eni potezi dosegljivi pokriti položaji. Do množice teh položajev pridemo na učinkovit način tako, da izhajamo vzvratno iz pokritih položajev v nepokrite. Pri tem nam ni treba upoštevati tistih pokritih položajev, iz katerih vse poteze vodijo le v ostale že pokrite položaje.

Ko dodamo novo pravilo, moramo uporabiti na obstoječi bazi položajev vsa že dodana pravila za označevanje še nepokritih položajev, dokler pravilo ne pokrije nobenega položaja več. Ta operacija čiščenja je potrebna zato, ker pri seznamu urejenih pravil lahko vsako pravilo uporabimo večkrat zaporedoma. Obravnava napačno in pravilno pokritih položajev se lahko izvede šele po končanem čiščenju.

Za generiranje pravil lahko uporabimo varianto iskanja v širino. Na začetku si izberemo več potencialnih pravil, kjer lahko uporabljamo ostale postopke induktivnega logičnega programiranja ali atributnega strojnega učenja na podlagi še dosegljivih nepokritih položajev. V naslednjih korakih pa poglabljanje iskanja pomeni ponovno uporabljanje že obstoječih pravil. Med poglabljanjem potem izbiramo najbolj perspektivno pravilo.

V tem kontekstu so smiselne hevristike, ki minimizirajo število napačno in maksimizirajo število pravilno pokritih položajev. Če napačno pokritih položajev ni, se izognemo njihovem preverjanju in posebni obravnavi. Nobeno

od pravil strategije v [2] nima napačno pokritih položajev.

Koristna hevristika za zmanjševanje kompleksnosti se nanaša na čiščenje. Namreč, nezaželeno je, da pri čiščenju uporabljamo tudi prejšnja in ne le trenutno pravilo. Zato lahko med čiščenjem zavrnemo vsa pravila, pri katerih lahko uporabimo tudi kako prejšnje pravilo, kar nam še dodatno oklešči število možnih pravil ter poveča razumljivost strategije. V [2] vsa pravila strategije ustrezajo temu pogoju.

Prav tako se nam v hevristiki splača minimizirati število popolnoma pokritih položajev, do katerih s pomočjo trenutne strategije zagotovo pridemo le iz pokritih položajev. Popolnoma pokritih položajev nam namreč ni več treba uporabljati za vzvratno pripravljanje seznama potencialno pokritih položajev po novem pravilu.

Sklepamo, da je bila pri človekovi pripravi strategije v [2] uporabljena med drugim tudi zgornja množica hevristik, ki so bistveno zmanjšale kompleksnost iskanja. Ob tem ima človek tudi bazo konceptov, ki izhajajo iz analognih problemov, in za katera se ve, da so uporabna.

5 Spekulativna izhodišča za nadaljnje delo

V tem razdelku bom obravnaval nekaj idej, ki so se mi porodile ob delu na tem problemu. Primeri, ki jih navajam, so le ilustracija konceptov in so presplošni, da bi bili neposredno uporabni. V bolj specifičnih okvirjih pa bi lahko služili kot izhodišča za iskanje človeku razumljivih strategij.

5.1 Monoliti

Strojno učenje je prezahteven problem, da bi še naprej iskali eno samo enostavno in genialno rešitev za vsak podproblem, temveč se moramo osredotočiti na integracijo velikega števila hevristik in pristopov. To lahko dosežemo tako, da pripravimo nekakšno osnovno okolje, v katerega se lahko hevristike in postopki vključujejo, se izbirajo in preizkušajo. Želimo si velikega števila drobnih postopkov, ki se lahko med seboj prosto povezujejo v enem samem sistemu. To bi morala biti prihodnost znanstvenega in inženirskega dela na področju strojnega učenja in ne nadaljnja gradnja monolitov, sestavljenih iz statično vpetega zaporedja rešitev, kar počne vsaka raziskovalna skupina posebej, ali še huje, vsak posameznik zase, ponavadi ponovno od samega začetka.

Seveda pa ta napor ni smiseln brez novih programskih jezikov, ki bodo omogočali, da postopki brez sprememb preživijo spremembe operacijskih sistemov, predstavitev podatkov, nedelujočih ostalih komponent, hroščev, neskončnih zank, itd. Delo na integraciji je namreč smiselno, le če je trajno.

5.2 Koncepti

Glavni izziv je odkrivanje novih konceptov. Tu imajo ljudje prednost, saj lahko uporabljajo množico že znanih analogij in drugačnih sfer razmišljanja. To nam postane jasno že s tem, da človek svoje koncepte poimenuje z relativno eno-

stavnimi in kratkimi imeni. Imena označujejo preizkušene in široko uporabne koncepte.

Ljudje relativno redko ustvarijo popolnoma nove besede in popolnoma nove koncepte. A ko so ustvarjeni, preizkušeni in široko uporabni, postanejo del naše orodjarne za razmišljanje.

V strojnem učenju bi ta koncept lahko uporabili tako, da si zapomnimo pogosto uporabljane rešitve in jih poskusimo na enostaven način specializirati. Šele, če do rešitve ne pridemo s kratkimi specializacijami, se lotimo daljših. Seveda tu ne gre za pravilnost, temveč za razumljivost in ekonomičnost predstavitve.

Da bi učinkovito uporabljali veliko množico bazičnih konceptov, potrebujemo ustrezne rešitve za to, na primer hierarhijo konceptov, potrebne pogoje za uporabo konceptov ter pojem konteksta. Pojem deljenja je specializacija pojma razmikanja in spremembe integritete, specializacije deljenja pa so na primer razbijanje, izločanje, rezanje, odcepljanje, razpolovljenje, osamitev, polarizacija, itd. Prav tako lahko stolpec ali vrstico 8 polj pri šahu obravnavamo kot črto, kar omogoči bolj kompakten opis in hitrejše delo. Da je nek koncept uporaben, obstaja nekaj potrebnih pogojev: delimo lahko le stvari, ki so sestavljene (zato deljenje uporabimo šele, ko smo dokazali sestavljenost). Če delimo z nečim, moramo tudi vedeti, s čim delimo. Kontekst pa pomeni na primer to, da v primeru strategije uporabimo koncepte, ki se nanašajo na strategijo in vojno, ne pa koncepte za področje medicine ali rudarstva.

5.3 Jezik in razumevanje

Jezik ne smemo obravnavati kot na izraz resnice, temveč kot na program, ki ljudem prihrani iskanje v širino pri razumevanju ideje. Človekovo interno razumevanje je bolj natančno od jezika samega, saj vsebuje podrobnosti, ki so z jezikom težko opisljive. Če me nekdo vpraša, zakaj ne grem ven, bom rekel, da je vroče. To ne pomeni, da je bila temperatura večja ali enaka 40 stopinj Celzija, temveč da je mojo odločitev mogoče razumeti z upoštevanjem vročine in ne razpoloženja, obveznosti ali česa drugega. Nesmiselno je poskušati enačiti jezik (tudi logiko) z razumevanjem.

Če želimo človeku dati korist z umetno analizo podatkov, zadošča le splošen oris in bistvena izhodišča, ne pa podrobnosti, ki jih lahko človek najde sam. Iskanje v globino je enostavno, ko končno najdemo dobro izhodišče z iskanjem v širino.

Ni pa hitrost dela z kompleksnejšimi koncepti edina prednost uporabe večjega števila osnovnih konceptov. Da bi ljudje razumeli rezultate, dobljene s strojnim učenjem, je bistveno, da računalnik uporablja koncepte, ki jih lahko opiše z naravnimi besedami. Zato bi moral računalnik imeti definirane podobne koncepte, kot jih poznamo ljudje.

6 Zaključek

Iz vidika razumljivosti je ocenjevanje poteznih ocen bolj smiselno kot učenje ocen položajev. Človeku je bolj razumljivo pravilo, ki primerja poteze med

seboj kot pa pravilo, ki opisuje položaje z isto oceno. Za primerjavo potez imamo namreč veliko več konceptov že v naravnem jeziku.

Problema smo se lotili iz smeri maksimizacije učinkovitosti, relativno enostavnega popravka strategije do polnosti in naknadnega preverjanja pravilnosti. Izkaže se, da iskalni postopki dajo strategijo, ki odlično deluje za nekatere partije, oziroma odseke partij. Se pa včasih izgubi, nekaterih uporabnih pravil pa ne dojame povsem (zmanjševanje prostora, trdnjava deli oba kralja).

Zagotavljanje pravilnosti in enostavnosti bi moralo biti v ospredju prihodnjih naporov v tej smeri. Če polnost in pravilnost nista zagotovljeni, je optimizacija iz vidika učinkovitosti nesmiselna. Z naraščanjem optimalnosti tudi zelo nezaželena kompleksnost strategij zelo hitro raste. Za nadaljnje poskuse z uporabo atributnega strojnega učenja v končnici KRK je nujno, da so vse generirane strategije pravilne in polne, strojno učenje pa lahko služi za izbiranje med strategijami ali njihovo poenostavitev. Generiranje strategij bi moralo iti od najkrajših še pravilnih proti bolj optimalnim.

Suboptimalnost je tudi kratkovidna. V primeru, da ne moremo najti optimalne strategije, moramo izbirati med več slabimi potezami. Vendar pri obravnavanju njihove slabosti ne moremo upoštevati le poteze same, ampak tudi slabost potez, ki bodo sledile. Temu problemu nismo našli ustreznega odgovora. Položaji in pravila z visoko suboptimalnostjo nam predstavljajo priložnost, da lažje najdemo boljše atribute, ki bodo uspešno razlikovali med dobrimi in slabimi pravili.

Nabor atributov bistveno omejuje prostor možnih strategij, zato te spominjajo tisto strategijo, za katero so bili določeni osnovni atributi. A kljub temu postopek učenja s suboptimalnostjo ne uspe najti vseh pravil osnovne strategije, še posebej tistih, ki so (dobro) suboptimalna med tem, ko strategija vleče naključne a omejene poteze, da bi našla izhodišče za nadaljevanje igre in ob tem preprečujejo, da bi se položaj bistveno poslabšal.

Literatura

[1] Bain M., *Learning Logical Exceptions in Chess.* PhD thesis, University of Strathclyde, 1994.

[2] Bratko I., *Prolog Programming for Artificial Intelligence,* Addison-Wesley, 1986.

[3] Bratko I., *Reševanje nekaterih kombinatoričnih problemov s hevrističnimi metodami,* Doktorska disertacija, Fakulteta za elektrotehniko, Ljubljana, 1978.

[4] Quinlan J.R., Learning efficient classification procedures and their application to chess end games. In R. Michalski, J. Carbonnel, and T. Mitchell, eds., *Machine Learning: An Artificial Intelligence Approach,* pp. 464–482. Tioga, Palo Alto, CA, 1983.

[5] Quinlan J.R., Induction of Logic Programs: FOIL and Related Systems, *New Generation Computing 13,* pp. 287–312, 1995.