računalniško generiranje popolnih strategij za šahovske končnice

m.gams
i.bratko

UDK 681.3:794.1

Institut J.Stefan in Fakulteta za elektrotehniko,
Univerza v Ljubljani, Ljubljana

V članku je opisan algoritem, ki v smeri od zadaj naprej zgradi prostor vseh možnih pozicij dane igre in za vsako pozicijo določi njeno vrednost. S tem algoritmom smo izračunali popolno strategijo za šahovsko končnico kralj in trdnjava proti kralju in skakaču, kar je zaradi kompleksnosti izračuna zahtevna programska naloga. Rezultati so zanimivi tudi s stališča same šahovske igre, ker je optimalna igra v tej končnici v splošnem že preveč težavna za šahovske mojstre. Ta eksperiment vodi k zaključku, da je na tak način možno v celoti reševati končnice z največ s pet do šest figurami, medtem ko postane od tu dalje kompleksnost izračuna že praktično nesprejemljiva in je potrebna uporaba metod umetne inteligence.

COMPUTER GENERATION OF COMPLETE STRATEGIES FOR CHESS END-GAMES. The article presents a backward-chaining algorithm for the generation of the space of all possible positions for a given game and for each position its value is computed. Using this algorithm, a complete strategy for the king and rook vs. king and knight chess ending was generated, which proved to be a programming task of considerable difficulty. The results are of interest also from the chess game point of view since optimal play in this ending is beyond chess-master's skill. A conclusion of the experiment is that endings with at most 5 or 6 pieces can still be completely solved in this way, while for more complex endings the computation becomes infeasable and from here on artificial intelligence methods are to be employed.

UVOD

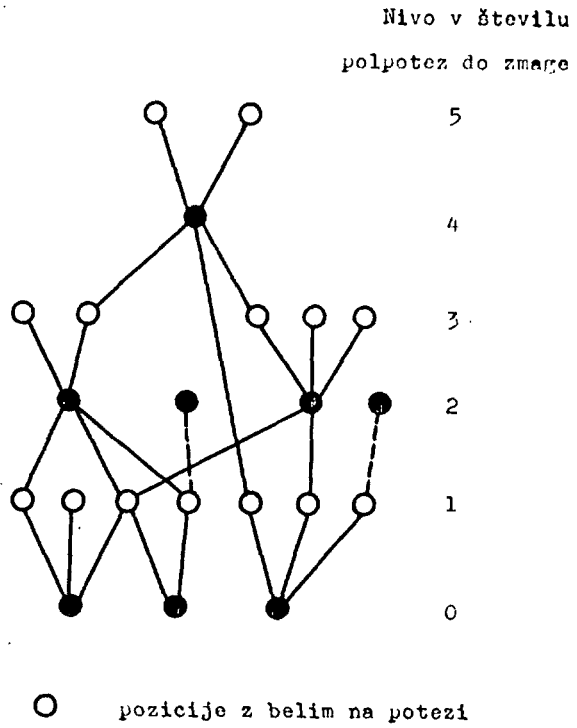
Znani so razmeroma enostavni algoritmi, ki bi načeloma popolnoma pravilno igrali šah. Mednje sodi npr. algoritem, ki preišče vsa možna nadaljevanja iz dane pozicije, dokler ne naleti na končne pozicije, tj. take, ki so dobljene, izgubljene ali remi po pravilih igre (npr. eden izmed kraljev v matu). Imenujmo tak algoritem "algoritem A". Hitro postane očitno, da algoritem A praktično ni izvedljiv zaradi svoje *časovne kompleksnosti*.

V povprečju je v vsaki šahovski poziciji možnih okrog 35 potez. Tako mora algoritem A v dani začetni poziciji, kjer je npr. na potezi beli, pregledati 35 potez belega; za vsako od nastalih pozicij mora upoštevati po 35 možnih odgovorov črnega itd. Tako mora za preiskovanje samo do globine ene cele poteze (tj. dveh polpotez, belega in črnega) pregledati okrog 1000 pozicij, za vsako naslednjo potezo v globino pa število pozicij naraste še za faktor 1000. Če optimistično ocenimo, da lahko računalnik generira po eno legalno potezo v eni mikrosekundi, potem bi potrebovali za izračun najboljše poteze v začetni šahovski poziciji po algoritmu A čas v velikostnem razredu, ki presega 10^{100} let (starost našega vesolja je približno 10^{10} let).

Algoritem A preiskuje možna nadaljevanja v smeri naprej od dane začetne pozicije proti končnim pozicijam. Drugi algoritem, ki tudi v načelu rešuje šahovsko igro, preiskuje prostor možnih pozicij v smeri nazaj. Imenujmo ga algoritem B. Algoritem B generira popolno strategijo npr. za belega tako, da generira najprej vse pozicije, ki so po definiciji dobljene za belega, npr. črni kralj v matu. Te pozicije so dobljene v 0 polpotezah (slika 1). Iz vseh teh pozicij poišče vzvratne poteze belega in tako dobi množico vseh pozicij, ki so dobljene za belega v eni polpotezi. Zatem poišče vse pozicije s črnim na potezi, v katerih vse možne poteze črnega vodijo v pozicije, dobljene za belega v eni polpotezi. Tako imamo množico vseh pozicij, dobljenih za belega v dveh polpotezah. Izhajajoč iz te množice lahko prejšnji postopek ponovimo in dobimo pozicije, dobljene v 3 polpotezah, 4 polpotezah itn. Kot končni rezultat algoritma B dobimo množico vseh dobljenih pozicij, pri čemer za vsako pozicijo vemo tudi število polpotez, potrebnih do zmage ob najboljši obrambi nasprotnika. Tako klasificirana množica možnih pozicij nam omogoča ne samo pravilno igro temveč tudi *optimalno igro, optimalno v tem smislu, da dobljeno pozicijo* privedemo do zmage v minimalnem številu potez.

Ker je *vseh možnih* šahovskih pozicij okrog 10^{44} (npr. Berliner 1978) in ker moramo pri izvajanju algoritma B hraniti vsaj vse dobljene pozicije, je jasno, da tudi algoritem B za celotno šahovsko igro ni izvedljiv. Možno pa je z algoritmom B rešiti končnice z manjšim številom figur. Tako je npr. v končnici s *tremi* figurami, npr. kralj in kmet proti kralju (kratko: končnica KPK), možnih manj kot 2×64^3 pozicij (ki se med seboj razlikujejo po položaju treh figur na šahovnici in po tem, kdo je na potezi). Zaradi simetrije lahko to število v končnici KPK reduciramo še za faktor 2 in tako dobimo problemski prostor z okrog 200.000 legalnimi pozicijami, ki ga je mogoče praktično obvladati z algoritmom B. Znana tovrstna rešitev te končnice je opisana v Clarke (1977). Če dodamo eno figuro, se velikost problemskega prostora poveča približno za faktor 64, kompleksnost algoritma B pa raste z večanjem števila figur še hitreje.

Kompleksnost algoritma A raste eksponencialno z globino iskanja v smeri naprej. Kompleksnost algoritma B pa raste eksponencialno s številom figur, prisotnih v končnici, ki jo *rešujemo. Jasno je, da postane zaradi eksponencialne rasti* tako kompleksnost algoritma A kot algoritma B praktično nesprejemljiva že za razmeroma majhno globino iskanja oziroma za razmeroma majhno število figur v končnici. Od teh dveh mejnih vrednosti naprej šahovske igre ne moremo več programirati s premočrtnimi eksaktnimi algoritmi, kot sta algoritma A in B, temveč z uporabo hevrističnih metod oziroma metod umetne inteligence.



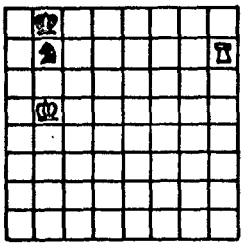
Sl. 1: Grafični prikaz delovanja algoritma B. Algoritem B generira pozicije od spodnjih nivojev navzgor

Postavljata se vprašanji:

(a) Do kakšne globine iskanja je algoritem A še praktično izvedljiv?
(b) Za kolikšno število figur v končnici je algoritem B še praktično izvedljiv?

Na vprašanje (a) dajejo odgovor izkušnje s turnirskimi šahovskimi programi. Trenutno najmočnejši in najhitrejši šahovski program, CHESS 4.7, preišče (če igra pod turnirskimi igralnimi pogoji, tj. po nekaj minut razmišljanja na potezo) vse variante do globine 8 ali 9 polpotez, odvisno od tipa pozicije, pri čemer teče na hitrem računalniku CYBER 176. Pri tem je algoritem A implementiran z znanim alfa-beta postopkom (npr. Knuth, Moore, 1975), ki je hitrejši od algoritma A, daje pa enak rezultat kot A. Izkušnje kažejo, da bi bilo tudi s precej hitrejšim računalnikom težko doseči bistveno večjo globino iskanja, npr. globino preko 10 polpotez, tj. 5 polnih potez.

Eksperiment, opisan v preostalem delu tega članka, pa daje odgovor na vprašanje b. V tem poskusu smo izračunali z algoritmom B popolno strategijo za končnico štirih figur: kralj in trdnjava proti kralju in skakaču (končnica KTKS). Rezultati tega izračuna, ki so zanimivi tudi s stališča same šahovske igre, predstavljajo pomembno orodje pri eksperimentiranju z metodami umetne inteligence, apliciranimi na to eksperimentalno okolje. Da mora biti pravilna strategija za to končnico generirana s pomočjo računalnika, potrjujejo rezultati psiholoških eksperimentov v Kopec, Niblett (1978), ki kažejo na to, da je popolnoma pravilna igra v tej na videz enostavni končnici že izven dometa šahovskih mojstrov. Celo obsežna obravnava te končnice v klasični Fineovi knjigi o šahovskih končnicah je polna netočnosti. Primer je na sliki 2.



Sl. 2: R. Fine (Basic Chess Endings, 1964), v skladu z analizami Bergerja, ocenjuje to pozicijo kot remi. Računalniško generirana strategija za končnico KTKS vsebuje za to pozicijo 15 potezno zmagovito varianto, ki se začenja z 1.Kc6.

Naš poskus kaže, skupaj z nekaterimi podobnimi znanimi rezultati, da je za končnice 4 figur še mogoče graditi popolne strategije z algoritmom B, da pa je sama programska izvedba tega algoritma zaradi časovnih omejitev zelo zahtevna. Če namreč želimo, da je čas izvajanja programa za rešitev celotne končnice v razumnih mejah, npr. nekaj ur, potem je treba najti učinkovito metodo za predstavitev pozicij v računalniku ter za organizacijo podatkovnih struktur na zunanjem pomnilniku. Npr. premočrtna, neustrezno strukturirana implementacija podatkovnih množic z datotekami z naključnim dostopom lahko poveča časovno kompleksnost programa za nekaj velikostnih razredov.

Če število figur v končnici povečamo samo za 1, postane izračun popolne strategije še neprimerno težji. Rešitev končnice kralj, trdnjava in kmet proti kralju in trdnjavi (Arlazarov, Futer, 1977) se lahko po programski zahtevnosti primerja z izdelavo prevajalnikov za programirne jezike. Končnice s 6 figurami pa so (razen v trivialnih primerih) verjetno že na meji dosegljivega pri današnjem stanju tehnologije.

ALGORITEM B

Spodnji algoritem zgradi množico pozicij, ki so dobljene za "nas" proti "njim" (ne za belega proti črnemu). V algoritmu so uporabljene naslednje kratice:

ttm - them to move, oni na potezi (nasprotnik na potezi, npr. črni na potezi)
utm - us to move, mi na potezi (npr. beli na potezi)
L - število polpotez do zmage ob optimalni igri obeh igralcev
n(L) - število pozicij z L polpotezami do zmage ob optimalni igri
poz(p) - število polpotez do zmage za pozicijo p
goal(p) - "končni predikat" za razpoznavanje pozicij, ki so po definiciji dobljene, npr. mat.

1. Inicializacija.
2. Za vse ttm pozicije označi vse nelegalne poteze.
3. Generiranje pozicij nivoja 0: vse ttm pozicije p, ki izpolnjujejo končni predikat goal(p), označi s "satisfied" in poz(p) = 0, n(0) = število pozicij s poz(p) = 0, L = 0.
4. Če je n(L) ≠ 0 in so še možne inverzne poteze nazaj iz pozicij nivoja L, nadaljuj, drugače končaj.
5. L = L+1, n(L) = 0.
6. Če je L sod, pojdi na 9, drugače nadaljuj s 7.
7. Generiraj utm nivo L iz ttm nivoja L-1: Za vsako ttm pozicijo p, označeno s "satisfied", naredi:

poišči vse predhodnike p (generiraj inverzne "us-move" poteze iz p) in za vsakega predhodnika q (na utm nivoju), če q ni označen s "satisfied", naredi:
1. označi q s "satisfied"
2. poz(q) = L
3. n(L) = n(L) + 1

8. Pojdi na 4.
9. Generiraj ttm nivo L iz utm nivoja L-1. Za vsako utm pozicijo p, označeno s "satisfied" in poz(p) = L-1 naredi:
poišči vse predhodnike p (generiraj inverzne "them - move" poteze iz p) in za vsakega naslednika q (na ttm nivoju), če q ni označeno s "satisfied", naredi:
1. označi ustrezno potezo iz q z "badmove"
2. če so vse poteze "badmove", potem
1. označi q s "satisfied"
2. poz(q) = L
3. n(L) = n(L+1)
10. Pojdi na 4.

Za izvedbo algoritma B potrebujemo naslednje osnovne podatkovne strukture:

UTMP: vektor utm pozicij dolžine N (= število vseh možnih pozicij). Za vsako pozicijo p: poz(p) (če je poz(p) večje kot $L_{max} = \max(L)$, potem p ni označeno s "satisfied"). Indeks pozicije v vektorju je koda pozicije p.

TTMP: vektor ttm pozicij dolžine N. Za vsako pozicijo p:
1 bit za oznako "satisfied"
M bitov za M možnih "them move" potez
(če je pozicija p označena s "satisfied", potem je v teh M bitih shranjeno poz(p)).

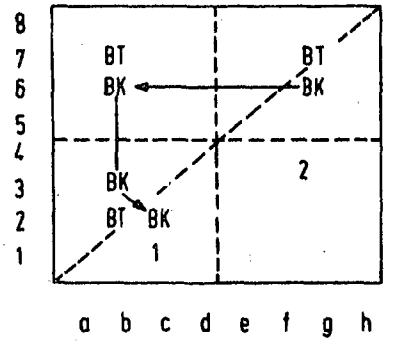
Za hitrejšo izvedbo algoritma običajno potrebujemo še:

UTMNEW: vektor bitov dolžine N. UTMNEW(koda(p)) = 1, če je bila utm pozicija p označena na predhodnem nivoju, to je poz(p) = L-1

TTMNEW: isto kot UTMNEW za ttm pozicije.

OCENITEV ŠTEVILA POZICIJ ZA KONČNICO KRALJ + TRDNJAVA : KRALJ + SKAKAČ

Ker imamo na šahovski deski (64 polj) 4 figure, je vseh možnosti 64 . 63 . 62 . 61, to je 15249024 pozicij. V tej oceni so zajete tudi pozicije, ki niso možne. Z upoštevanjem simetrije lahko v prvem zamahu zmanjšamo število pozicij na 10 . 63 . 62 . 61 = 2382660. Šahovsko desko razrežemo na 8 trikotnikov in izbrano figuro s transformacijami prevrtimo v izbrani trikotnik velikosti 10 polj, kot je to razvidno s slike 3. Pri tem je BT bela trdnjava, BK beli kralj, CK črni kralj in CS črni skakač.



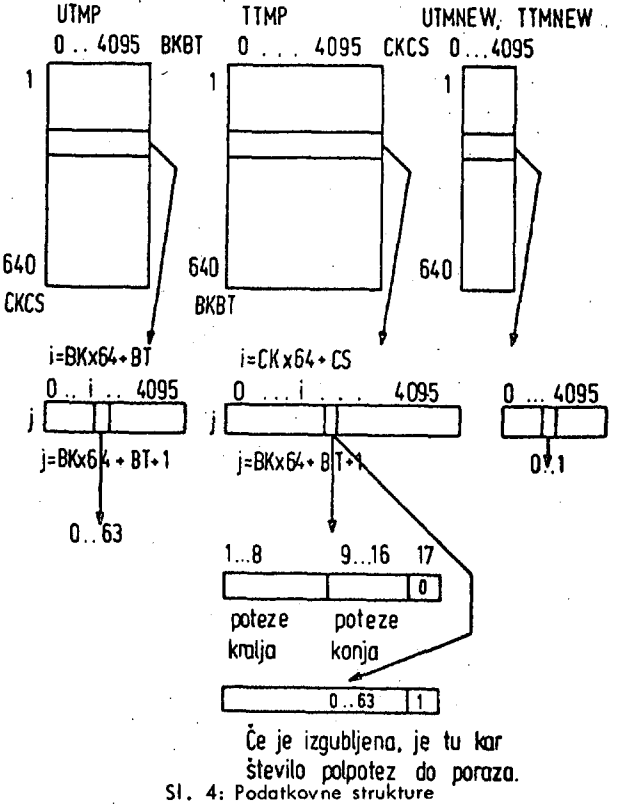
Sl. 3: Zmanjševanje števila pozicij z upoštevanjem simetrije. Preslikavanja BK (belega kralja) v trikotnik 1. BT (bela trdnjava) je zraven, da lažje vidimo, kako potekajo preslikave preko osi simetrije

Figure so urejene po prioriteti, recimo BK, BT, CK, CS. Če je beli kralj na diagonali (a1 - d4), lahko po prioriteti drugo figuro preslikamo v trikotnik 1+2. Če so vse prioritetnejše figure na diagonali, lahko zavrtimo pozicijo preko diagonale (a1 - h8) tako, da po prioriteti najpomembnejša figura, ki ni na diagonali, pristane v trikotniku 1+2. S tem lahko še bolj zmanjšamo število pozicij na eno osmino od 15249024 = 1906128. Seveda je treba razlikovati še v tem, kdo je na potezi, tako da je dejansko število pozicij dvakrat večje.

IMPLEMENTACIJA ALGORITMA B

Pri izvajanju algoritma B je treba za vsako možno pozicijo hraniti določeno množino informacij (po 6 bitov za utm pozicije oziroma po 17 bitov za ttm pozicije, kot je razvidno iz nadaljevanja). Tako je celotni potrebni pomnilni prostor pri obdelavi končnice KTKS velikosti okrog 60 milijonov bitov.

V grobem lahko časovno kompleksnost algoritma B ocenimo takole: za vsako dobljeno pozicijo moramo generirati vse možne inverzne poteze, tj. za končnico KTKS v velikostnem razredu 30 milijonov. Tolikokrat je potem potrebno tudi ažurirati podatke v datotekah skupne velikosti 60 milijonov bitov. Zato premočrtna organizacija teh podatkov, tako da bi bilo pri izvedbi algoritma B potrebnih 30 milijonov naključnih dostopov, praktično ni sprejemljiva. V naslednjih odstavkih so opisane podatkovne strukture, ki omogočajo, da ažuriramo te podatke povečini sekvenčno.



Sl. 4: Podatkovne strukture

Na sliki 4 so osnovne podatkovne strukture. To so datoteke dolžine 640 okenc, okence pa je polje 4096 elementov. Indeks okenca dobimo iz položaja dveh najbolj prioritetnih figur. Položaj figure je dan z zaporedno številko polja na katerem stoji figura, pri čemer so polja oštevilčena od 0 do 63. Kratice BK, BT, CK in CS v nadaljevanju pomenijo po vrsti položaj belega kralja, bele trdnjave, črnega kralja in črnega skakača. Npr. UTMP je datoteka, katere okence i = CK x 64 + CS + 1 ($0 \leq CK \leq 9$, v trikotniku 1). Indeks polja v okencu je določen s figurama manj prioritetne barve, za

UTMP je to BK x 64 + BT. Element polja je za UTMP kar število polpotez do zmage ali 63, če beli iz te pozicije ne more izsiliti zmage. Za TTMP je element polja v primeru, če je pozicija izgubljena (bit št. 17 je v tem primeru 1), število polpotez do poraza.

Dokler z algoritmom še nismo ugotovili, da je pozicija izgubljena, je 17. bit nič, element polja pa sestavlja še 16 bitov, prvih 8 za poteze kralja, drugih 8 za poteze konja. Če je i-ta poteza nič, vodi v za črnega neizgubljeno pozicijo. Če je 1, je ali nelegalna ali vodi v pozicijo, kjer lahko beli z optimalno igro izsili zmago. Element polja v okencu za UTMNEW in TTMNEW je kar en bit in je 1 v primeru, če je ta pozicija dobljena za belega ali izgubljena za črnega v L polpotezah za tekoči L in nič drugače (L je oznaka nivoja).

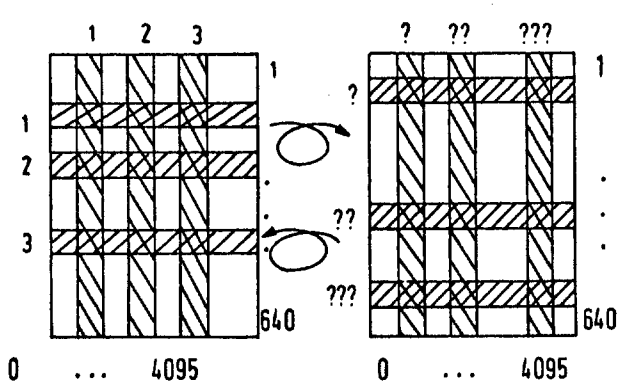
Datoteka UTMP zavzema 2 621 440 x 6 bitov = 15 728 640 bitov. Datoteka TTMP ima 2 621 440 x 17 bitov = 44 564 480 bitov, obe datoteki skupaj približno 60 milijonov bitov. Hitrost računalnika CYBER 172 je okrog milijon tristo tisoč operacij na sekundo. Če za pregled vsake poteze v vsaki poziciji porabimo čas stotih operacij, bo to približno 16 x 2 milijona x 100 operacij, to je približno ena ura računalniškega časa. V resnici je program, napisan v Pascalu, porabil nekaj več kot 3 ure.

Tako urejene podatkovne strukture imajo več dobrih lastnosti, predvsem pa:

1. V primarnem pomnilniku imamo samo eno okence vsake datoteke.
2. Če v neki poziciji delamo poteze s figurami manj prioritetne barve, so vse tako dobljene pozicije še vedno v okencu, saj nam indeks okenca določata bolj prioritetni figuri. Kolikor nam je znano, drugi avtorji ne uporabljajo take ureditve, najbrž zato, ker niso uspeli rešiti problema prehoda iz UTMP v TTMP in obratno. To omogočata pomožni datoteki UTMNEW in TTMNEW.

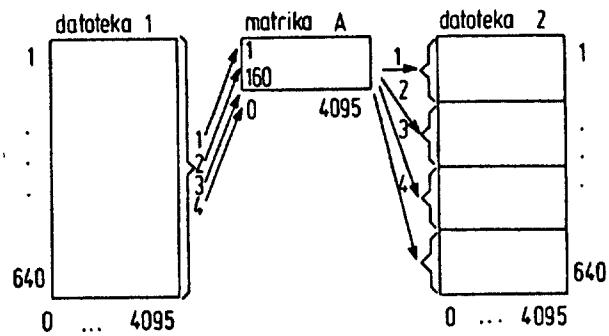
Ko naredimo vse poteze z manj prioritetno barvo in tako zgradimo nivo L, za vsako pozicijo nivoja L sproti vpišemo "1" v malo datoteko UTMNEW ali TTMNEW. Nato pozicije nivoja L prevrtimo iz ene v drugo malo datoteko.

Kot je razvidno s slike 5, se določene vrstice male datoteke preslikajo v določene stolpce druge male datoteke, določeni stolpci pa v vrstice, vendar se vrstni red znotraj stolpcev oziroma vrstic ne ohranja. Indeks vrstice je določen z bolj prioritetnima figurama iste barve, recimo CK * 64 + CS + 1, indeks stolpca pa z manj prioritetnima figurama, torej BK * 64 + BT.



Sl. 5: Vrtenje malih datotek

Vidimo, da so vse datoteke urejene tako, da jih procesiramo sekvenčno, torej bistveno hitreje, kot če bi jih morali naključno. Ostal pa nam je problem vrtenja malih datotek pri prehodu iz nivoja L na nivo L+1. Ker sta datoteki preveliki za primarni pomnilnik, si pomagamo z vmesno matriko A, kot je to razvidno s slike 6. Velikost te matrike je četrtina velikosti male datoteke in gre v celoti v primarni pomnilnik.



Sl. 6: Programska realizacija vrtenja malih datotek

Matrika bitov A je v primarnem pomnilniku, zato je procesiranje z naključnim dostopom hitro, medtem ko sta obe mali datoteki, 1 in 2, v sekundarnem pomnilniku in ju procesiramo sekvenčno. Matrika A ustreza četrtini male datoteke, zato moramo štirikrat sekvenčno "prečesati" malo datoteko 1. Pri tem z masko gledamo samo stolpce, ki vsebujejo pozicije, ki se lahko preslikajo v matriko A. Če v okencu male datoteke opazimo 1, potem zamenjamo prioriteto figur in enko z novim indeksom prepišemo v matriko A. Ko smo s prvo masko preleteli malo datoteko 1, se v matriki A nahajajo vsi stolpci datoteke 1, ki se bodo prepisali v prvo četrtino male datoteke 2. Matriko A po vrsticah prenesemo v prvo četrtino datoteke 2 in nadaljujemo postopek z masko 2.

Za uspešno realizacijo projekta je bil poudarek tudi na učinkovitosti podprogramov, ki se pogosto izvajajo. Poglejmo si primer enega izmed njih. Pri tem je npr. BKX koordinata X polja, ki ga zaseda beli kralj.

```
Function  NICSAMB : boolean;
(* funkcija je true, če črni ne daje belemu šaha s konjem *)
begin
     p := true;
     j := abs(BKX - CSX);
        if  j < 3  then
           if  j ≠ 0  then
              if  j = 1  then  p := abs(BKY - CSY) ≠ 2
                         else  p := abs(BKY - CSY) ≠ i ;
     NICSAMB := p
end; (* NICSAMB *)
```

Zato, da je gornja funkcija čimbolj učinkovita, so bile pri vrstnem redu testov upoštevane verjetnosti, da so pripadajoči pogoji izpolnjeni.

REZULTATI

Po končanem generiranju celotnega problemskega prostora za končnico KTKS smo dobili naslednje statistične rezultate. Vseh legalnih pozicij za belega na potezi je 1 347 906, za črnega pa 1 567 222. Narejena je bila tudi statistika po številu polpotez do zmage belega ob optimalni igri obeh nasprotnikov.

Beli na potezi		Črni na potezi	
št. polpotez	št. pozicij	št. polpotez	št. pozicij
1	378518	0	33156
3	95450	2	54539
5	46269	4	14839
7	30749	6	15242
9	20055	8	10563
11	15071	10	9294
13	11740	12	8493
15	9495	14	7721
17	8562	16	7664
19	7415	18	7917
21	6308	20	7014
23	5356	22	6085
25	4133	24	5129
27	3356	26	3776
29	2290	28	2921
31	1621	30	2025
33	1333	32	1580
35	1046	34	1390
37	727	36	1006
39	556	38	707
41	458	40	544
43	373	42	396
45	302	44	286
47	178	46	178
49	111	48	135
51	18	50	41
53	2	52	13
		54	1

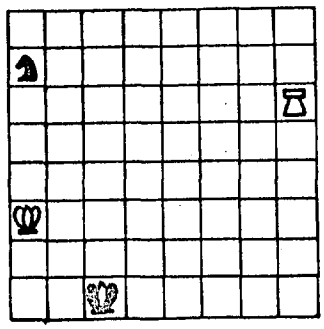
Skupaj	
Beli	Črni
651 492	221 191

Statistika je presenetljiva. Za bele z večjim številom polpotez do zmage število dobljenih pozicij lepo pada, zato pa ima statistika črnih dva skoka. Za 2 polpotezi do poraza je število pozicij večje kot za po definiciji izgubljene pozicije. Podoben skok opazimo pri 4-6 in 16-18. Druga presenetljiva ugotovitev je, da je pri majhnem številu polpotez do zmage belega število belih pozicij z L polpotezami večje kot število črnih z L-1 polpotezami, kar je razumljivo, vendar se razmerje spremeni v korist črnih od L = 18 - 19 naprej in ostane tako do konca razen za L = 44 - 45, kjer je belih več kot črnih, in za L = 46-47, kjer je število pozicij za bele in črne enako.

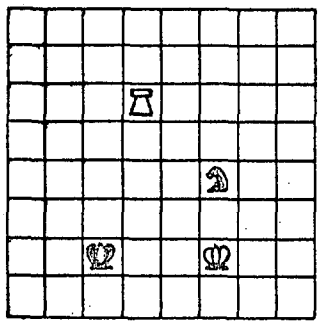
Pri statistiki so rezultati dobljeni z upoštevanjem skrčitve števila pozicij, zato je resnično število pozicij osemkrat večje.

Vidimo, da obstaja ena sama pozicija, kjer se črni na potezi lahko upira še 27 potez (= 54 polpotez). Poglejmo si optimalno odigrano partijo iz te pozicije. Pri tem se v oklepajih pojavljajo ekvivalentne variante (poteze, ki vodijo do zmage belega v istem številu polpotez). Pravilna igra je tu tako zahtevna, da že sodi v problemski šah. Poskusi Kopeca in Nibletta (1978) so pokazali, da šahovski mojstri praktično nimajo nobenih možnosti, da bi teoretično dobljeno izhodiščno pozicijo privedli do zmage pod turnirskimi igralnimi pogoji.

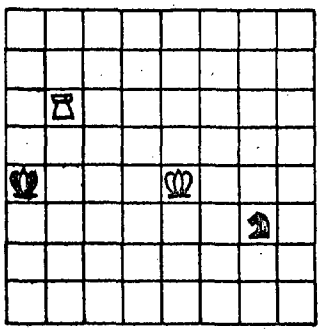
Začetni položaj je na spodnjem diagramu.



1. Sb5
Obe črni figuri sta dokaj blizu središča, pozicija izgleda prej remi kot poraz za črnega.
2. Kb4 Sd4, 3. Kc3 Se2, 4. Kd3 Sf4,
5. Ke3 Sg2!, 6. Kf2 Sf4,
Dvoboj belega kralja in črnega konja je končan, črni konj ni uspel priti pod zaščito svojega kralja.
7. Td6 Kc2



Kralj skuša priti v aktivnejši položaj, toda bela trdnjava zdaj ločuje obe črni figuri.
8. Ke3 Sg2, 9. Kf3 Sh4, 10. Ke2! Sf5,
11. Tc6 Kb3, 12. Kd3 Kb4
Po manevriranju ostaneta črni figuri še naprej ločeni, vendar sta še blizu središča.
13. Te6 Kb3!, 14. Tb6 Ka4, 15. Ke4 Sg3
(15. Kc4 ..)



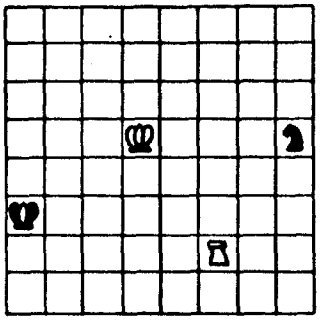
Beli sili črnega v čedalje slabše pozicije, črni figuri sta še bolj ločeni in že odrinjeni iz središča.
16. Kd5 Se2 (16. Kd4 ..) (16. .. Ka3),
17. Kc4 Ka3, 18. Tb1 Sf4 (18. .. Tb3 ..),
19. Tg1 Se6 (19. Te1 ..), 20. Tg6 Sf4,
21. Tf6 Sh5 (21. Tb3 ..), 22. Tf2 Sg7 (22. Tf3..),
23. Kd5 Sh5 (23. .. Kb3, b4, a4).

Če bi poskušali s kakšnim preprostim algoritmom tipa A za preiskovanje v smeri naprej optimalno odigrati takšno partijo, bi za to potrebovali približno 1000 krat toliko časa, kolikor je stara Zemlja.

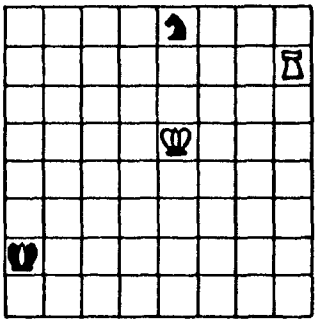
V eni potezi dobimo do 352 novih pozicij iz ene stare. Če namesto 352 vzamemo faktor 10 in za generacijo ene pozicije porabimo 10^{-6} sekunde, bo to

$$\frac{10^{27} \cdot 10^{-6}\,s}{10^{18}\,s} = 1000$$

Pri tem je 10^{18} s starost Zemlje.



Obe beli figuri se odpravljata na lov na črnega konja.
24. Ke5 Sg7,
25. Th2 Se8 (25. Tf7, f8 ..) (25. .. Ka4, b3, b4)
26. Th7 Ka2 (26. .. Kb2, b3, b4, a4)



Črni konj je očitno izgubljen.
27. Te7 Sg7 (27. .. Ka1, a3, b1, b2, b3, Sc7, d6, f6)
28. Tg7:

LITERATURA:

1. Arlazarov, V.L., Futer, A.V., Computer analysis of a rook end-game, v Machine Intelligence 9 (Ed. D.Michie), v tisku.
2. Berliner, H.J., Computer Chess, Nature, Vol. 274, 24. Aug. 1978, 745-748.
3. Clarke, M.R.B., A quantitative study of king and against king, v Advances in Computer Chess 1, 108-118 (Ed. M.R.B. Clarke), Edinburgh: University Press, 1977.
4. Knuth, D.E., Moore, R.W., An analysis of alpha-beta pruning, Artificial Intelligence, 6, 293-326, 1975.
5. Kopec, D., Niblett, T., How difficult is the king and rook vs. king and knight ending?, v Advances in Computer Chess 2, (ed. M.R.B. Clarke), v tisku.