Iskanje matov pri progresivnem šahu
1. seminarska naloga pri predmetu Algoritmi

Ivan Antešić[1]

Fakulteta za računalništvo in informatiko, Univerza v Ljubljani,Večna pot 113, Slovenija
ia6382@student.uni-lj.si

Keywords: A* · Hevristika · Progresivni šah · Algoritmi.

1 Uvod

V okviru prve seminarske naloge sem razvil algoritem za iskanje matov pri progresivnem šahu. Progresivni šah je ena od različic šaha pri kateri ima vsak naslednji igralec na vrsti odigra za eno potezo daljše zaporedje kot prejšni. Pri tem velja je še, da lahko nasprotniku postavimo šah ali mat izključno v zadnji potezi, ki jo imamo na voljo.

Zaradi kombinatorične zahtevnosti problema je pri razvoju iskalnih algoritmov potrebno določiti dobre hevristike, ki usmerjajo iskanje in s tem skrajšajo čas iskanja končne rešitve.

Pri razvoju algoritma sem si pomagal z magistrsko nalogo[1] in člankom[2] Vita Janka in Mateja Guida. V delih so opisane glavne ideje algoritma, ki sem jih nato prilagodil in implementiral.

Algoritem sem napisal v razvojnem okolju IntelliJ v jeziku Java. Uporabil sem tudi knjižnjico ProgressiveChess.jar, ki vsebuje pravila progresivnega šaha. Delovanje sem testiral na strežniku, ki program požene na 60 primerih.

2 Rešitev

Za iskalni algoritem sem uporabil A*, ki preiskuje vozlišča (v našem primeru stanja šahovnic, ki so posledica določene poteze) in med njimi poišče pot od začetka do konca (mata). V vsakem koraku algoritma vzame vozlišče z najmanjšo ceno iz prioritetne vrste in ga preišče, tako da njegovim otrokom izračuna ceno ter jih doda v vrsto. Šahovsko FEN notacijo preiskanega vozlišča nato shrani v zgoščeno množico (hashMap), zato da v prihodnosti ne bi preiskal istega vozlišča.

Ceno vozlišča ocenimo z optimističnimi (vedno ocenijo manj kot je dejaska cena) hevristikami, ki poskrbijo, da algoritem iz množic vozlišč izbira tista, ki prej vodijo do cilja. V nadaljevanju so opisane hevristike, ki sem jih uporabil pri iskanju matov.

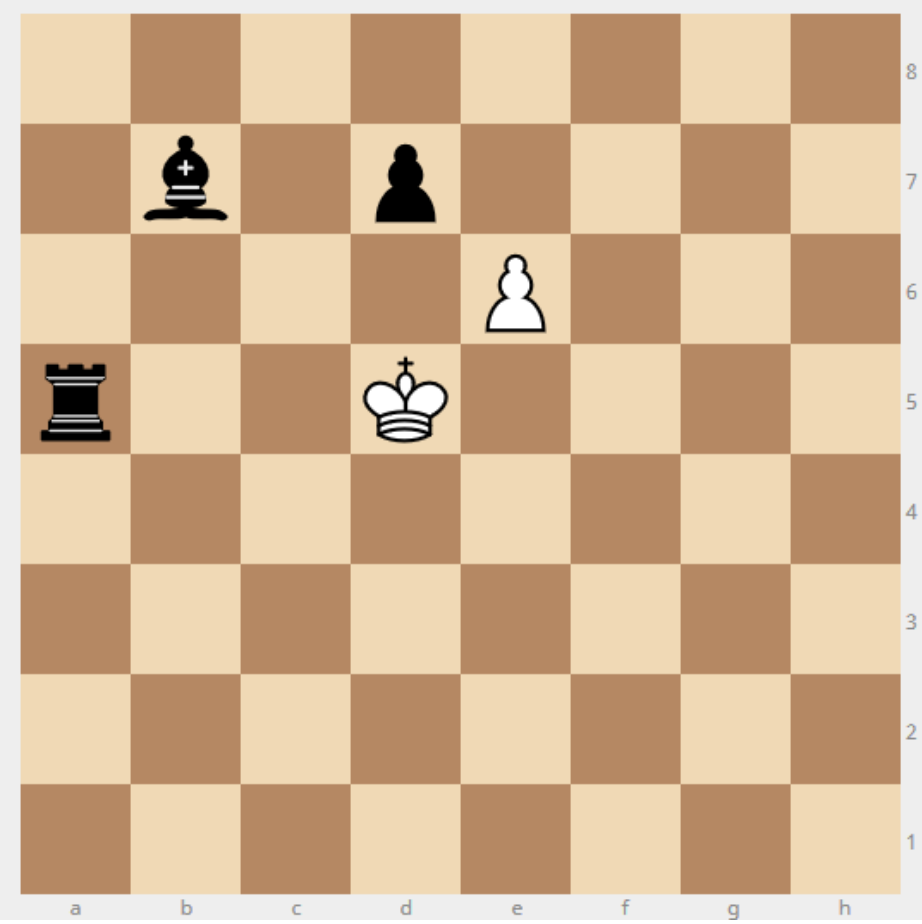

Fig. 1. Za zgornji primer šahovnice je vrednost manhattanske hevristike 9, vrednost pokritosti pa -6. Ker večja pokritost ugodno vpliva na izbiro vozlišča je v računanju predstavljena kot negativno število.

2.1 Manhattan

Manhattanska je preprosta in naivna hevristika, ki se izračuna zelo hitro. Sešteje manhattanske razdalje od vseh igralčevih figur do nasprotnikovega kralja. Med testiranjem se je izkazalo, da deluje bolje če pri tem upošteva tudi kmete. Na sliki 1 je izračunana vrednost razdalje za prikazano šahovnico. Hevristika teži k temu, da vse figure poriva v smeri proti kralju.

Čeprav na splošno rešitev najde počasneje kot pokritost je zmožna pod časovno omejitvijo 20 sekund najti tudi 4 mate, kjer imamo na voljo 6 potez in enega z 9 potezami. Skupno sem z uporabo Manhattana rešil 34/60 testnih

primerov v času 794 sekund. Za rešene primere je povprečno algoritem porabil 11 sekund.

2.2 Pokritost

Pokritost je definirana kot število neposrednih polj okoli kralj (ter polje samega kralja), ki jih pokrivajo igralčeve figure. Na sliki 1 je izračunana pokritost za prikazano šahovnico. Z testiranjem sem ugotovil, da deluje bolje če pri tem ne upošteva kmetov.

Ker je pokrivanje teh polj odločilnega pomena v matu je ta hevristika bolj usmerjena kot manhattanska razdalja. Izkaže se, da deluje bolje ko imamo na voljo 4 ali 5 potez, kasneje pa se preveč razširi. Prav tako ima pri večini primerov na začetku vrednost 0, kar sprevrže algoritem v sikanje najglobjega, dokler ne pridejo figure v pozicijo prekrivanja. Posledično reši 28/60 primerov v 899 sekundah.

2.3 Končna hevristika

Da bi izboljšal delovanje algoritma, sem definiral nekaj dodatnih manjših hevristik, ki sem jih nato združil skupaj. Konča hevristika je izračunana po sledeči formuli:

$0.1 * Manhattan + 0.9 * pokritost - 0.8 * globina$ in dodatno -0.5 če poteza vsebuje promocijo v kraljico.

Ta rešitev se je izkazala kot najboljša, ker reši 36/60 primerov v 743 sekundah. Rešene primere izvede v povprečno 7 sekundah.

Kombinacija V želji, da bi pridobil dobre lastnosti Manhattana in pokritosti ter omilil njihovi slabosti, sem uporabil obe naenkrat. Utežil sem jih po naslednji formuli: $0.1 * Manhattan + 0.9 * pokritost$.

Računanje manhattanske razdalje je hitro, kljub temu pa dodatna majhna vrednost pomaga pokritosti, ko ima le ta vrednost 0. V takih primerih pomika vse figure proti kralju, dokler ne začnejo pokrivati matna polja. Nato vpliv manhattana postane manjši, pokritost pa začne natančo usmerjati poteze.

Globina Pri A* ponavadi poleg hevristike upoštevamo tudi globino poti. V ta namen sem hevristiki prištel še število že narejenih potez, ki so vodilo v trenuno vozlišče. Izazalo se je, da je rezultat v tem primeru slabši (22/60 rešenih primerov v 1042 sekundah) in boljši, če to vrednost odštejemo. Posledica je širše iskanje bolj plitkih vozlišč. Sklepam, da je to zaradi pravil progresivnega šaha, ki dovoljujejo mat šele v zadnji dovoljeni potezi - osredotočanje takoj na globje poteze ni dobrobitno.

Promoviranje Da bi nagradil poteze, ki vodijo do promoviranja kmeta, sem hevristiko utežil glede na najmanjšo pot, ki jo ima igralčev kmet nasprotnikovega roba. To se ni izkazalo za dobro nadgradnjo, saj hevristika prekomerno premika kmete proti robu. V primeru, da je rob nedostopen zaradi nasprotnikove figure nima nobenega vpliva.

Zato sem rajši preprostejše nagradil potezo v kateri se kmet promovira v kraljico, ki je najboljša možna promocija.

3 Zaključek

Vpliv hevristik na izvajanje iskalnega algoritma je velik. S kombiniranjem različnih hevristik je možno izboljšati njihove slabosti in uporabiti dobre lastnosti. Z eksperimentiranjem sem določil uteži posameznih parametrov hevristike, ter dosegel zadovoljiv rezultat.

V primerjavi z magistrskim delom seveda moj algoritem deluje zelo počasneje. Za izboljšanje bi bilo potrebno še več eksperimentiranja z različnimi, novimi hevristikami.

Literatura

1. Janko, V., Guid, M.: Razvoj programa za igranje 1-2-3 šaha: magistrsko delo. V. Janko (2015)
2. Janko, V., Guid, M.: A program for progressive chess. Theoretical Computer Science **644**, 76 – 91 (2016). https://doi.org/https://doi.org/10.1016/j.tcs.2016.06.028, `http://www.sciencedirect.com/science/article/pii/S0304397516302730`, recent Advances in Computer Games