Učenje računalniškega šaha z uporabo algoritma diferencialne evolucije

Borko Bošković, Sašo Greiner, Janez Brest, Viljem Žumer
Univerza v Mariboru
Fakulteta za elektrotehniko računalništvo in informatiko
Inštitut za računalništvo
Smetanova ulica 17, 2000 Maribor, Slovenija
borko.boskovic@uni-mb.si

Abstract

The big challenge of artificial intelligence is to create system that can learn from own experiences. One of possible way is using evolutionary approaches. Chess programs in those approaches present individuals in population. During the evolution, individuals learning and improving their configurations or efficiency. In this paper we are focus on a "Differential Evolution" which gives better results than other evolutionary approaches.

1 Uvod

Šahovski programi so danes zelo močni igralci, enakopravni so celo najboljšim igralcem sveta. Tako se postavlja vprašanje, kako lahko te programe še izboljšamo. S konvencionalnim inženirskim pristopom tega ne moramo narediti. Razlog temu je omejeno šahovsko znanje razvijalcev. Ker pa imamo na razpolago računalniške vire, lahko šahovske programe izboljšamo s pomočjo strojnega učenja. Ta metoda omogoča učenje programov na osnovi lastnih izkušenj. Programi tako dosegajo stopnjo šahovskih mojstrov, brez uporabe znanja ekspertov.

Najbolj pomembna komponenta šahovskih programov je ocenitvena funkcija. Le ta vsebuje vso šahovsko znanje programov. Glavni problem pri razvoju močnega šahovskega programa predstavlja načrtovanje in implementacija ocenitvene funkcije. Ocenitvena funkcija lahko vsebuje mnogo različnih izrazov, ki jih je potrebno združiti v celoto. To opravilo je težko in opravimo ga lahko s pomočjo strojnega učenja oz. iskanja optimalne ocenitvene funkcije. V ta namen lahko uporabimo različne algoritme, kot so npr. vzpenjanje na hrib, simulirano ohlajanje, učenje s povratno informacijo in evolucijski algoritmi.

Ena prvih metod strojnega učenja šahovskih programov je uporaba algoritma "Temporal Differences Learning" (TDL). To metodo je predstavil Samuel [1]. Kasneje jo je razširil in formaliziral Sutton [2] v njegovem algoritmu TD(λ). Mnogo raziskovalcev je nato algoritem TDL apliciralo v svoje programe [3, 4, 5]. Eden najbolj uspešnih šahovskih programov, v katerega je bil algoritem TDL apliciran, je "NeuroChess".

Drugo metodo predstavljajo evolucijski algoritmi. Algoritmi strojnega učenja temeljijo na principu Darwinove evolucije [11]. Ti algoritmi lahko prilagodijo šahovske programe, da igrajo z ratingom šahovskega mojstra. Tukaj na zmogljivosti pridobimo preko serije iger v evolucijskem procesu znotraj učnega okolja, ki se odziva na lastnosti posameznikov (učencev) v populaciji. Serije iger se uporabljajo za ugotavljanje zmogljivosti posameznikov oz. šahovskih programov. Graham Kendall in Glenn Whiwell [7] sta predstavila evolucijski pristop za uglaševanje ocenitvene funkcije z iterativno metodo selekcije članov populacije. V [6] so predstavili uspešno načrtovan šahovski program, ki se je skozi evolucijski pristop učil na lastnih izkušnjah. Tako se je program skozi evolucijo izboljšal za skoraj 400 rating točk in dosegel moč mojstrske stopnje.

Tudi mi smo se odločili, da bomo šahovski program učili s pomočjo evolucijskega pristopa. Izbrali smo algoritem diferencialne evolucije (DE - "Differential Evolution") [9, 10]. Ta algoritem se zelo uspešno uporablja pri raznih optimizacijskih postopkih, npr. pri iskanju globalnih optimumov večdimenzionalnih matematičnih funkcij. Učinkovitost algoritma DE smo preizkusili z uporabo našega šahovskega programa [8, 12] in preproste ocenitvene funkcije, ki jo je zasnoval Shannon. Rezultati eksperimenta so pokazali, da je učenje z algoritmom DE učinkovitejše kot v primeru drugih evolucijskih algoritmov.

V drugem poglavju predstavljamo šahovski program, ki ga bomo učili. Algoritem DE predstavljamo v tretjem poglavju. Četrto poglavje opisuje prilagoditev algoritma DE strojnemu učenju ter eksperiment učenja in dobljene rezultate. Kratek zaključek opravljenega dela in nadalnje delo podajamo v petem

poglavju.

2 Šahovski program

Šahovski program, ki ga bomo učili, vsebuje naslednje osnovne komponente: predstavitev igre, generator potez, iskalne algoritme, transpozicijsko tabelo, ocenitveno funkcijo in otvoritveno knjižnico. Predstavitev igre vključuje bitno predstavitev pozicij (Bitboard) in potez. Generator potez temelji na MVV/LVA (Most Valuable Victim / Last Valuable Attacker) [8, 12] strategiji. Osnovna ideja tega generatorja potez je, da generira seznam potez, ki na začetku vsebuje poteze jemanja z največjo vrednostjo žrtve in najmanjšo vrednostjo napadalca, npr. kmet jemlje damo.

Iskalni algoritmi temeljijo na alfa-beta algoritmu. Za časovno omejeno iskanje se uporablja iterativno poglabljanje, ki je izboljšano z aspiracijskim iskanjem. Aspiracijsko iskanje preiskuje iskalni prostor z določenim iskalnim oknom in tako reducira čas iskanja. V terminalnih vozliščih iskalnega drevesa smo uporabili iskanje mirovanja (Quiescence search), ki v primeru dinamičnih poziciji (vsebujejo poteze jemanja in promocije) nadaljuje iskanje do t.i. statičnih pozicij (ne vsebujejo potez jemanj in promocij). V primeru pozicij, kjer je igralec na potezi v šahu, iskanje nadaljujemo z alfa-beta algoritmom in globino iskanja 1. Tako iskanje mirovanja zna odkriti mat pozicije. Transpozicijska tabela nam omogoča, da se izogibamo nepotrebnemu večkratnemu preiskovanju ene in iste pozicije.

Programu smo dodali še otvoritveno knjižnico, ki programu omogoča v začetni fazi igre izbiro najboljših potez. Program vsebuje še enostavno ocenitveno funkcijo, ki jo je zasnoval Shannon. Prikazuje jo enačba 1 in vsebuje informacije o materialu in mobilnosti figur.

$$ocenitev = \sum_{y=0}^{6} \Big[W_y(N_{y,beli} - N_{y,črni}) + W_m(M_{y,beli} - M_{y,črni}) \Big] \quad (1)$$

V prikazani enačbi N predstavlja število, W utež in M mobilnost določenega tipa figur, W_m pa predstavlja utež mobilnosti figur.

3 Algoritem DE

S pomočjo algoritma DE smo poskušali poiskati optimalne parametre v prej opisani ocenitveni funkciji. DE je enostaven in učinkovit algoritem za globalno optimizacijo. Hitro in enostavno ga lahko implementiramo in uporabljamo, saj vsebuje le nekaj kontrolnih parametrov. DE je algoritem, ki v populaciji vsebuje NP D-dimenzionalnih vektorjev oz. posameznikov.

$$\overrightarrow{x}_{i,G} \quad , \quad i = 1, 2, 3, ..., NP \quad (2)$$

V (2) G predstavlja eno generacijo. V vsaki generaciji imamo eno populacijo. V našem primeru posameznike populacije predstavljajo šahovski programi oz. vektorji uteži ocenitvenih funkcij. Osnovna ideja algoritma DE je v kreiranju novih posameznikov. Novi posamezniki se kreirajo z dodajanjem obteženih razlik med dvema posameznikoma tretjemu posamezniku populacije. Če je novi posameznik boljši v primerjavi s posameznikom iz populacije, posameznika iz populacije zamenjamo z novim posameznikom.

Uporabili smo DE/rand/1/bin strategijo algoritma DE [9, 10]. Ta strategija se sestoji iz inicializacije, mutacije, križanja in selekcije. Inicializacija naključno izbira vektorje začetne populacije. Mutacija za vsak vektor $\overrightarrow{x}_{i,G}$ ustvari vektor $\overrightarrow{v}$ na naslednji način:

$$\overrightarrow{v}_{i,G+1} = \overrightarrow{x}_{r_1,G} + F(\overrightarrow{x}_{r_2,G} - \overrightarrow{x}_{r_3,G}), \quad (3)$$

r_1, r_2 in r_3 so naključno izbrani različni indeksi, ki so različni od i in na intervalu $[1, NP]$. Ti indeksi se znova naključno izbirajo za vsak naslednji indeks i. F je realno število na intervalu $[0, 2]$, ki ga določi uporabnik in predstavlja faktor skaliranja vektorja $(\overrightarrow{x}_{r_2,G} - \overrightarrow{x}_{r_3,G})$. Mutiran vektor nato križamo na naslednji način:

$$\overrightarrow{u}_{i,G+1} = (u_{1i,G+1}, u_{2i,G+1}, u_{3i,G+1}, ..., u_{Di,G+1})$$
$$u_{ji,G+1} = \begin{cases} u_{ji,G+1}, & r(j) \leq CR \vee j = rn(i) \\ x_{ji}, & r(j) > CR \wedge j \neq rn(i) \end{cases} \quad (4)$$

j predstavlja uniformno naključno izbran indeks na intervalu $[1, D]$. CR je konstanta križanja, ki jo določi uporabnik na intervalu $[0, 1]$. $rn(i)$ je naključno izbran indeks na intervalu $[1, D]$. Dobljen vektor nato primerjamo z i-tim vektorjem populacije in na osnovi selekcije izberemo posameznika, ki bo preživel v naslednjo generacijo. Selekcija temelji na naslednjem pravilu:

$$\overrightarrow{x}_{i,G+1} = \begin{cases} \overrightarrow{u}_{i,G+1}, & f(\overrightarrow{u}_{i,G+1}) > f(\overrightarrow{x}_{i,G}) \\ \overrightarrow{x}_{i,G}, & f(\overrightarrow{u}_{i,G+1}) \leq f(\overrightarrow{x}_{i,G}) \end{cases} \quad (5)$$

Funkcija $f(\overrightarrow{x}_{i,G})$ je ocenitvena funkcija vektorja $\overrightarrow{x}_{i,G}$. Če je ocena novega vektorja boljša od ocene vektorja iz populacije, vektor iz populacije zamenjamo z novim vektorjem. Drugače populacija ostane nespremenjena.

4 Eksperiment

S predstavljeno strategijo algoritma DE smo učili predstavljen šahovski program. V ta namen smo implementirali algoritem DE, ki vsebuje selekcijo prilagojeno strojnemu učenju.

Osnovni problem, ki se pojavi pri selekciji, je ugotavljanje, kateri program oz. posameznik v populaciji je boljši. V ta namen nastavimo uteži (vektorji iz populacije) dvema programoma. Ta programa odigrata sodo število iger tako, da je en program polovico iger beli in drugo polovico črni igralec. Tako oba programa odigrata enako število iger, v katerih imata prednost belega. Oba programa z igrami zbirata točke. Za zmago program dobi 2 točki, za remi 1 točko in 0 točk za poraz. Glede na število zbranih točk ugotavljamo, kateri program je boljši in po pravilu selekcije v naslednjo generacijo preživi boljši posameznik. V primeru neodločenega rezultata preživi posameznik iz populacije. Ta posameznik je odigral več iger in verjetno predstavlja boljšega igralca. Programa lahko dvoboj tudi predčasno končata. To se zgodi v primeru, kadar eden od programov več ne more doseči zmage ali neodločenega rezultata.

Opisan pristop učenja z algoritmom DE smo uporabili pri eksperimentu učenja našega šahovskega programa. Program smo zasnovali tako, da smo uporabili ocenitveno funkcijo, ki jo prikazuje enačba 1. Ta enačba vsebuje uteži (vrednosti) figur in utež mobilnosti. Dejanske vrednosti uteži so naslednje:

W_{kmet}	$= 100$	$W_{trdnjava}$	$= 500$
$W_{skakač}$	$= 300$	W_{dama}	$= 900$
W_{lovec}	$= 330$	$W_{mobilnost}$	$= 10$

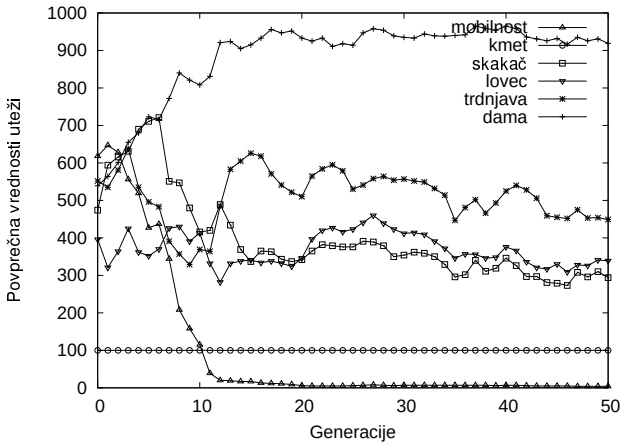
V eksperimentu smo uporabili začetno populacijo, ki je bila izbrana z uniformnim naključnim generatorjem na intervalu [0-1000] za vse uteži razen uteži kmeta. Utež kmeta smo fiksirali na vrednost 100. Velikost populacije (NP) je bila 20. Pri selekciji je bil uporabljen dvoboj s 4 igrami in preiskovanjem do globine 3. Kontrolni parameter F je bil nastavljen na vrednost 0.5 in CR na 0.9. Povprečne vrednosti uteži začetne populacije so bile naslednje:

W_{kmet}	$= 100$	$W_{trdnjava}$	$= 552.9$
$W_{skakač}$	$= 474.35$	W_{dama}	$= 539.75$
W_{lovec}	$= 396.45$	$W_{mobilnost}$	$= 618$

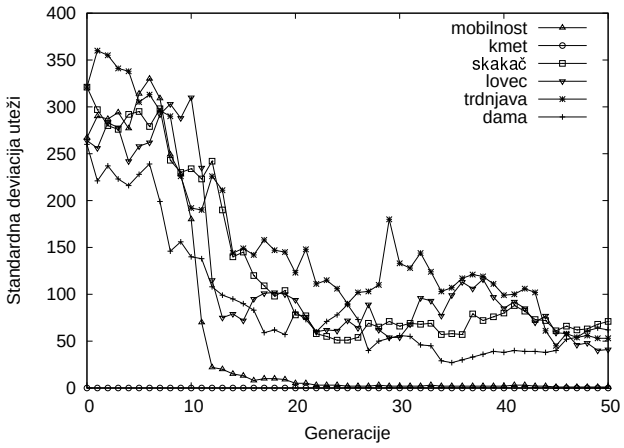
Standardne deviacije uteži so znašale:

σ_{kmet}	$= 0$	$\sigma_{trdnjava}$	$= 321.214$
$\sigma_{skakač}$	$= 321.626$	σ_{dama}	$= 260.143$
σ_{lovec}	$= 264.45$	$\sigma_{mobilnost}$	$= 267.447$

Eksperiment je trajal 50 generacij v katerih je bilo odigrano 3070 iger. Povprečno vrednost in standardno deviacijo populacije skozi generacije



Slika 1: Povprečne vrednosti



Slika 2: Standardna deviacija

prikazujeta diagrama 1 in 2. Na koncu eksperimenta smo dobili populacijo z naslednjimi povprečnimi vrednostmi uteži:

W_{kmet}	$= 100$	$W_{trdnjava}$	$= 454.4$
$W_{skakač}$	$= 310.4$	W_{dama}	$= 931.45$
W_{lovec}	$= 341.4$	$W_{mobilnost}$	$= 4.8$

Standardna deviacija pa se je zmanjšala na naslednje vrednosti:

σ_{kmet}	$= 0$	$\sigma_{trdnjava}$	$= 53.19$
$\sigma_{skakač}$	$= 68.34$	σ_{dama}	$= 64.68$
σ_{lovec}	$= 40.53$	$\sigma_{mobilnost}$	$= 1.79$

Rezultati eksperimenta kažejo, da so posamezniki v zadnji generaciji bolj podobni drug drugemu, kot je to bilo v začetni populaciji in so se približali dejanskim vrednostim. Opisan eksperiment smo ponovili tudi s klasičnim evolucijskim algoritmom [6], pri katerem je bilo učenje znatno počasnejše. Tudi glede na rezultate v [7] lahko ugotovimo, da je učenje z DE algoritmom primerljivo z evolucijskim pristopom z uporabo

semena. Če pa primerjamo število odigranih iger, se je DE s pomočjo naše selekcije izkazal kot učinkovitejši. Algoritme DE smo testirali še z različno globino iskanja in številom iger pri selekciji. V našem eksperimentu se je izkazalo, da ta dva parametra ne vplivata na učinkovitost učenja.

5 Zaključek

V članku smo predstavili metodo strojnega učenja šahovskega programa s pomočjo algoritma DE. V algoritmu DE smo uporabili selekcijo na osnovi sodega števila odigranih iger. Glede na rezultate iz literature in naših testov smo ugotovili, da algoritem DE lahko učinkovito uporabimo pri učenju šahovskih programov.

Izvedli smo eksperiment s pomočjo našega šahovskega programa in preproste ocenitvene funkcije, ki je vsebovala uteži materiala in mobilnosti. Vrednosti teh uteži so v evolucijskem procesu hitro dosegle določene vrednosti, ki pa se niso dosti razlikovale od dejanskih vrednosti. Tako lahko ugotovimo, da se je naš program skozi proces evolucije algoritma DE naučil razmerja vrednosti med figurami in mobilnostjo.

V nadaljevanju našega dela želimo dodati ocenitveni funkciji nevronsko mrežo in jo učiti. Tako bomo brez ekspertnega šahovskega znanja poskušali implementirati program, ki se bo lahko kosal z mojstri šahovske igre.

Opisan evolucijski pristop lahko uporabimo tudi za reševanje drugih problemov, za katere rešitve ne poznamo ali jih delno poznamo. Tako lahko iščemo novo znanje, ki ga lahko združimo z obstoječim znanjem. Evolucijski proces dodatno omogoča paralelno reševanje problemov ter tako naredi evolucijske algoritme časovno učinkovitejše. Minimalno, kar pa je potrebno, je primerjava dveh posameznikov v populaciji in ugotavljanje, kdo je boljši. Na ta način evolucijsko strojno učenje lahko doseže znanje ekspertov in ga tudi dopolni.

Literatura

[1] Samuel A. L. *Some Studies in Machine Learning Using the Game of Checkers.* IBM J. Res. Dev., Vol. 3(3) 1959.

[2] Sutton R. *Learning to Predict by the Method of Temporal Differences.* Machine Learning 3, pp. 9-44, 1988.

[3] Baxter J., Tridgell A. end Weaver L. *KnightCap: A chess program that learns by combining TD(λ) with game tree search.* Proceedings of the Fifteenth International Conference on Machine Learning, pp 28-36, July 1998.

[4] Thrun S. *Learning to Play the Game of Chess.* Advances in Neural Information Processing Systems (NIPS) 7, 1995.

[5] Tesauro G. J. *TD-Gammon, a self-teaching backgammon program, achieves master-level play.* Neural Computation (6), pp. 9-44, 1994

[6] David B. Fogel, Timothy J. Hays, Sarah L. Hahn and James Quon *A Self-Learning Evolutionary Chess Program.* Proceedings of the IEEE, Vol. 92, No. 12, pp. 1947-1954, December 2004.

[7] Graham Kendall and Glenn Whitwell *An Evolutionary Approach for the Tuning of a Chess Evaluation Function using Population Dynamics.* Proceedings of the 2001 IEEE Congress on Evolutionary Computation Seoul, Korea, May 2001.

[8] Bošković B., Greiner S., Brest J. and Žumer V. *The Representation of Chess Game.* proceedings of the 27th International Conference on Information Technology Interfaces, Croatia, June 2005.

[9] Storn R. and Price K., *Differential Evolution - A Simple and Efficient Adaptive Scheme for Global Optimization over Continuous Spaces.* Technical Report TR-95-012, Berkeley, CA, 1995.

[10] Storn R. and Price K., *Differential Evolution - A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces.* Journal of Global Optimization, vol. 11, pp. 341-359, 1997.

[11] Pollack J., Blair A., and Land M. *Coevolution of a Backgammon Player.* Proceedings of the Fifth Artificial Life Conference, Nara, Japan 1996.

[12] Borko Bošković. *Implementacija računalniškega šaha.* Fakulteta za elektrotehniko, računalništvo in informatiko. Diplomsko delo univerzitetnega študija, Maribor, 2004.