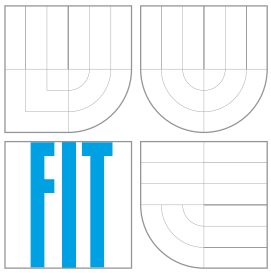
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

POUŽITÍ EVOLUČNÍHO ALGORITMU VE HŘE ŠACHY

EVOLUTIONARY ALGORITHM USED IN A CHESS GAME

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — ANDREJ URMINSKÝ
AUTHOR

VEDOUCÍ PRÁCE — Ing. ZBYŠEK GAJDA
SUPERVISOR

BRNO 2007

Abstrakt

Táto práca sa zaoberá návrhom evolučného algoritmu pre umelú inteligenciu v hre šach. K dosiahnutiu tohto cieľa je použitý tzv. genetický algoritmus. Pre implementáciu algoritmu a grafického rozhrania bol využitý programovací jazyk Java vo vývojovom prostredí Eclipse.

Klíčová slova

evolučný algoritmus, Java, šach, minimax, alfa-beta

Abstract

This thesis deals with a design of an evolutionary algorithm for an artificial intelligence in a chess game. This is accomplished by use of so called genetic algorithms. Java programming language and Eclipse, an open development platform, were used for implementation of this algorithm and the graphical user interface.

Keywords

evolutionary algorithm, Java, chess, minimax, alpha-beta

Citace

Andrej Urminský: Použití evolučního algoritmu ve hře šachy, bakalářská práce, Brno, FIT VUT v Brně, 2007

Použití evolučního algoritmu ve hře šachy

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Zbyška Gajdy.

.....
Andrej Urminský
13. května 2007

Poděkování

Rád by som poďakoval Ing. Zbyškovi Gajdovi za odbornú pomoc, za pomoc pri tvorbe tejto práce a za poskytnutú literatúru k tejto problematike.

Obsah

Kapitola 1

Úvod

Keď sa objavili prvé počítače, očakávali sa od nich veľkolepé výsledky - nie iba zo strany vedcov, ktorí ich skonštruovali, ale aj zo strany širokého publika, ktoré bolo novými mysliacimi strojmi fascinované. Mnoho špekulácií sa však uberalo nesprávnym smerom. Počítačová technológia síce urobila obrovský pokrok, ale nie v tých oblastiach, kde to experti predpokladali.

To isté platí pre počítače a šach. Bolo veľmi vzrušujúce, keď začali prvé počítače hrať šach. Prvýkrát sa vedci dočkali toho, že počítač robí niečo, čo u človeka vyžaduje inteligenciu. Bol to pre nich celkom jasný príklad inteligentného chovania stroja. Ich nadšenie bolo tak veľké, že niektorí z nich predpovedali blízke víťazstvo počítača nad majstrom sveta. Toto proroctvo sa však vyplnilo až o skoro pol storočia neskôr.

Počítačoví odborníci urobili vo svojich úvahách dve chyby. Za prvé precenili rýchlosť, s ktorou sa budú rozvíjať programovacie techniky. Učiace sa stroje, ktoré by samé korigovali svoj postup, sa nedali zrealizovať tak jednoducho, ako si to mnohí mysleli.

A za druhé, mnoho odborníkov podcenilo zložitosť šachovej hry. Naučiť počítač pohybovať figúrkami a vyvinúť metódy, s ktorých pomocou by hral šach na úrovni amatéra, bolo pomerne jednoduché. Ale napriek intenzívnemu výskumu v tejto oblasti sa doteraz nikomu nepodarilo predložiť koncepciu, ako sa dá pomocou počíťača dosiahnuť najvyššej dokonalosti ľudskej šachovej hry [3].

1.1 Cieľ a štruktúra práce

Ciele tejto bakalárskej práce sú: V prvom rade bolo potrebné sa zoznámiť s metódami z oboru umelej inteligencie, ktoré sa používajú v ťahových (doskových) hrách. Potom navrhnúť evolučný algoritmus pre použitie v hre šach. Ďalším krokom bola implementácia navrhnutého algoritmu spolu s implementáciou grafického rozhrania pre interakciu evolučného algoritmu s užívateľom/ hráčom. Posledným krokom práce je porovnanie navrhnutého a implementovaného algoritmu s inými produktami.

V kapitole 2 je uvedená história šachových programov, kapitola 3 oboznamuje čitateľa o algoritmoch využívajúcich sa v súčasných šachových programoch. Teória o evolučných algoritmoch sa nachádza v kapitole 4. V kapitole 5 som popísal navrhnutý a implementovaný evolučný algoritmus. Implementácia pomocou jazyka Java a vývojového prostredia Eclipse a dosiahnuté výsledky sú popísané v kapitolách 6 a 7. Pohľad na iné spôsoby implementácie nájdeme v kapitole 8. Predposlednou kapitolou 9 je Záver, kde je celá práca zhodnotená.

Kapitola 2

História šachových programov

2.1 Ohodnocovacia funkcia

Vedci v oblasti umelej inteligencie sa vždy zaoberali vývojom šachových programov. Claude Shannon bol jeden z prvých vedcov, ktorý navrhol počítačový program, ktorý by vedel hrať šach. Zvolil si ohodnocovaciu funkciu, pomocou ktorej by program mohol rozhodnúť o relatívnej hodnote rozličných konfigurácií figúriek na šachovnici. Pojem ohodnocovacia funkcia bol odvtedy súčasťou každého šachového programu. V bežnej ohodnocovacej funkcii nechýba posudzovanie materiálovej výhody, formácia pešiakov, pozícia figúriek, mobilita, uväznenie, útoky apod. Shannon poznamenal, že k najlepšiemu ťahu sa dá dopracovať minimálne dvomi spôsobmi, aj keď obe metódy sa dajú kombinovať: (1) Prehľadávanie do zadanej hĺbky ťahov a potom použitie algoritmu minimax alebo (2) selektívne prehľadávať ramená stromu hry do rozličnej hĺbky. Druhá metóda je výhodnejšia, pretože zabraňuje prehľadávaniu ramien (a tým zbytočným plytvaním času), v ktorých by sa vykonalo jeden alebo viac zlých ťahov. Táto metóda, neskôr pomenovaná alfa-beta, sa implementuje v takmer každom súčasnom šachovom programe [1].

2.2 Prvé pokusy o vytvorenie šachových programov

Prvý fungujúci šachový program bol vytvorený v roku 1956. V roku 1958 sa prvýkrát v šachovom programe použil algoritmus alfa-beta a v roku 1967 program nazývaný Machack VI zaznamenal prvú výhru nad klubovým hráčom. Program sa stal čestným členom Šachovej Federácie Spojených Štátov ohodnotený na 1640 ELO [1]. Machack VI používal prehľadávanie stavového priestoru do hĺbky minimálne 9 polťahov [1].

2.3 Vytváranie špecializovaného hardvéru

V 80-tych rokoch sa začal robiť dôraz na vytváranie hardvéru, ktorý je aplikačne špecifický, aby sa uľahčilo prehľadávanie veľkého množstva možných pozícií a urýchlilo vypočítavanie ohodnocovacou funkciou. Hans J. Berliner vytvoril Hitech, 64-procesorový systém. Fenghsiung Hsu priniesol ešte výkonnejší čip a jeho program, teraz známy ako Deep Thought, rýchlo prekonal Hitech. Deep Thought dokázal prehľadávať stavový priestor do hĺbky 10 polťahov a stal sa prvým programom, ktorý porazil jedného zo svetových majstrov, Benta

[1] medzinárodný systém merania výkonu v šachu [3]

Larsena. V roku 1989, Deep Thought, vtedy hodnotený na 2754 ELO, hral štvorhrový zápas so Škótskym šachovým šampiónom Davidom Levym. Levy prehral 4:0.

Vrchol šachovej umelej inteligencie priniesol v máji 1997 systém navrhnutý a skonštruovaný firmou IBM, nazvaný Deep Blue (následník Deep Thought). Deep Blue hral proti šachovému majstrovi sveta Garrymu Kasparovovi a Deep Blue ho porazil v dvoch hrách, raz prehral a trikrát s Kasparovom remizoval. Pod "kapotou" Deep Blue sa skrývalo 32 paralelných procesorov, čo umožňovalo prehľadávanie 200 miliónov šachových pozícií za sekundu a dosiahnutie hĺbky prehľadávania 12 polťahov [1].

Kapitola 3

Šachové algoritmy

Medzi najznámejšie a najpoužívanejšie algoritmy, ktoré sa používajú v šachu (ale aj rôznych iných stolných hrách player vs player, ako napríklad piškvorky, GO atď.) patria minimax a alfa-beta [8].

3.1 Šach a matematika

Existuje jedno obrovské číslo, ktoré je všeobecne známe a súvisí s šachom. Podľa legendy bol vynálezca šachovej hry istý Sissa ibn Dahir, veľkovezír indického vládcu Shirmana. Keď chcel kráľ svojho vezíra za peknú novú hru odmeniť, žiadal tento iba trochu pšenice: na prvé pole šachovnice jedno zrnko, na druhé dve zrnká, na tretie štyri atď. - na každé ďalšie pole dvojnásobok než na predchádzajúce. Kráľ sa síce spočiatku pre túto neprimeranú skromnosť hneval, ale nakoniec povolil. Skoro ale zistil, že v celom svojom kráľovstve ani toľko pšenice nemá. Sissovi totiž malo pripadnúť 18446744073709551615 zrniek ($1+2+2^2+2^3+...+2^{63}$). To pri dnešných výnosoch zhruba zodpovedá celosvetovej produkcii za ďalších 2000 rokov. Toto číslo je však smiešne malé, keď začneme hovoriť o kombinatorike šachovej hry.

V priemernej šachovej hre sa odohrá asi 40 ťahov, než jeden z hráčov zloží zbrane alebo sa obaja dohodnú na remíze. Celkom teda urobia 80 polťahov, to je 40 ťahov bieleho a 40 ťahov čierneho hráča. V bežnej šachovej partii má hráč, ktorý je na ťahu, k dispozícii asi 40 možných ťahov, z ktorých si musí jeden vybrať (toto číslo obdržíme, ak vyhodnotíme a spriemerujeme veľké množstvo náhodne vybraných pozícií). Ak chce počítač pre každý z týchto možných ťahov preskúmať tiež všetky možné protiťahy partnera, potom musí uvažovať $40 * 40 = 1600$ pokračovaní. Inými slovami: z priemernej pozície môže vzniknúť po dvoch polťahoch asi 1600 rôznych pozícií. Ako to pokračuje, vidieť v tabuľke 3.1 [3].

3.2 Porovnanie algoritmov minimax a alfa-beta

Alfa-beta algoritmus je mierne modifikovaný minimax, zabraňuje prehľadávať ťahy, ktoré nie sú potrebné ohodnotiť, a to tým, že prestane ohodnocovať ťah, keď nájde aspoň jednu možnosť, ktorá dokáže, že tento ťah je určite horší, ako predchádzajúci ohodnotený. V prípade minumaxu ide o úplné prehľadávanie stavového priestoru do zadanej hĺbky, zatiaľ čo u alfa-bety o "čiastočne úplné" prehľadávanie stavového priestoru do zadanej hĺbky. Čiastočne úplné preto, lebo účinnosť alfa-bety závisí v rozhodujúcej miere na poradí, v ktorom sa ťahy preverujú. V najhoršom prípade teda môže nastať prípad, keď sa alfa-beta "degraduje" na minimax. V tomto prípade by algoritmus alfa-beta mal zložitosť

Polťahy	Počet možných operácií	
1	40	
2	1600	
3	64 000	
4	2,6 miliónov	
5	102 miliónov	
6	4,1 miliardy	...vek človeka v sekundách
7	164 miliárd	
8	6,5 biliónov (10^{12})	
9	262 biliónov	
10	10 biliárd (10^{15})	
11	419 biliárd	...vek vesmíru v sekundách
12	17 triliónov (10^{18})	...Sissove pšeničné zrnká
13	671 triliónov	
14	$2,7 * 10^{22}$	
-	-	
-	-	
-	-	
49	$3,2 * 10^{78}$	...počet elementárnych
50	$1,3 * 10^{80}$	častíc vo vesmíre
-	-	
-	-	
-	-	
80	$1,5 * 10^{128}$	

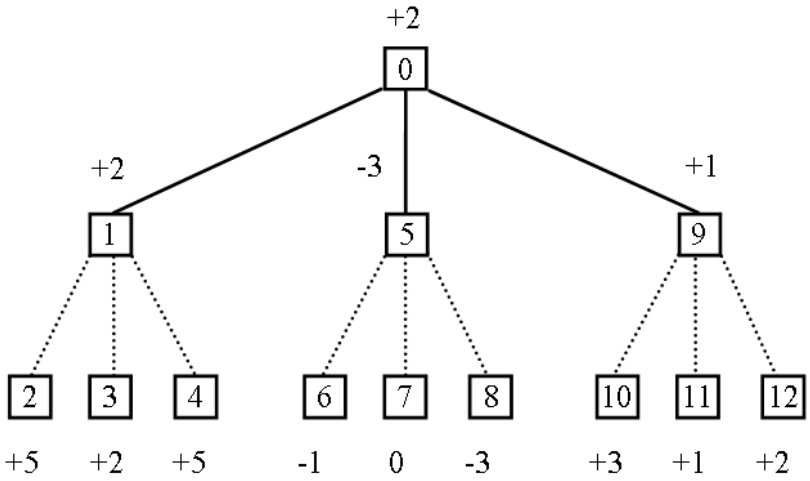
Tabuľka 3.1: Exponenciálna explózia pri šachu

rovnakú, ako minimax, t.j. x^y, kde x je počet možností v jednom ťahu (v našom prípade 40, zmienené v kapitole 3.1) a y vyjadruje počet polťahov, hĺbku prehľadávaného priestoru. Matematická analýza ukazuje, že pri optimálnom využití alfa-bety sa namiesto x pozícií prehľadáva iba $\sqrt{x}$. V praxi sa dá docieliť úspor rádovo $5 * \sqrt{x}$ [3].

Pri správnom poradí prepočtu ťahov sa v rovnakom čase s metódou alfa-beta dostaneme dvakrát hlbšie, než s minimaxom. Dôležité je, že zložitosť alfa-beta metódy silne závisí na poradí ťahov. To, do akej miery sa priblížime dvojnásobnej hĺbke minimaxu, je predovšetkým vecou heuristík pre triedenie ťahov [8].

3.3 Minimax

Minimax je metóda rozhodovacej teórie, ktorá slúži na minimalizovanie maximálne možnej straty. Jednoduchá verzia algoritmu sa používa v hrách, ako sú napríklad piškvorky, kde každý hráč môže vyhrať, prehrať alebo remízovať. Základom algoritmu minimax je rekurzívna funkcia, ktorá sa zavolá pre aktuálny stav hry (koreňový uzol AND/OR grafu) a hráča A. Táto funkcia vracia ohodnotenie uzlu a pre hráča A aj ťah k uzlu s maximálnym ohodnotením, t.j. ťah, ktorý je v danom stave hry pre tohto hráča najvýhodnejší. Funkcia minimax predpokladá, že je zadaná maximálna hĺbka prehľadávania (počet skúmaných ťahov).



Obrázok 3.1: Minimaxový vyhľadávací strom. Celky s plými čiarami označujú OR graf, časti s bodkovanými čiarami AND graf.

Príklad vybrania najvhodnejšieho ťahu môžeme vidieť na obrázku 3.1. Počítač vychádza z okamžitej pozície na šachovnici (0) a prevedie v pamäti prvý z troch možných ťahov. Vytvorí tým pozíciu (1), v ktorej má protivník na výber tri ťahy. Jeden z nich sa taktiež prevedie v pamäti počítača, čím sa získa prvá koncová pozícia (2). Tá sa ocení a pridelí sa jej hodnota +5, tzn. že po prevedení oboch ťahov by vznikla pozícia, v ktorej by mal počítač na šachovnici materiálnu prevahu rovnajúcu sa jednej veži.

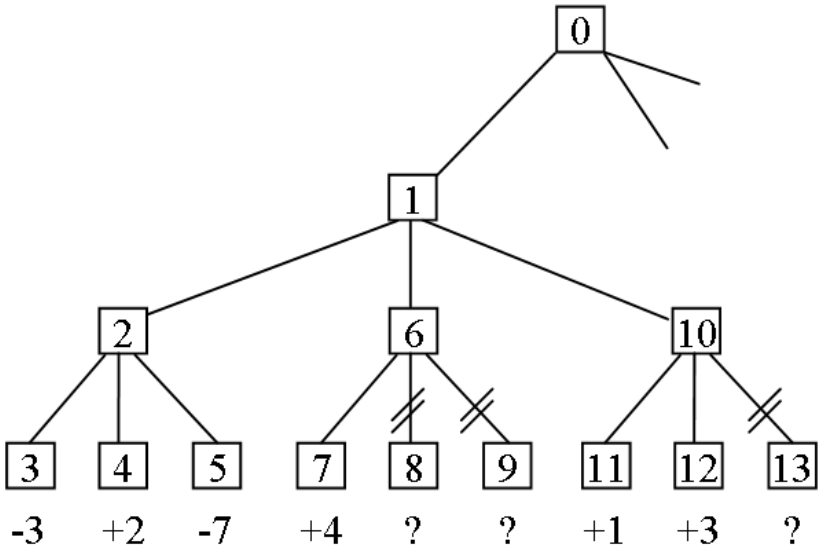
Teraz sa počítač vráti do pozície (1), tzn. vráti prvý ťah protivníka a prevedie druhý ťah protivníka. Ocení novú koncovú pozíciu (3), vráti sa späť do (1) a prevedie posledný protivníkov ťah. Tým vznikne pozícia (4). Všetky tri koncové pozície sa ocenia a počítač sa opäť vráti do východzej pozície (0).

Tomuto prvému ťahu pripíše počítač hodnotu +2, teda najhoršiu hodnotu, s ktorou sa po protivníkovych ťahoch stretol. Časť stromu, v ktorej sa vyberajú najhoršie hodnoty (kde je na ťahu súper), sa nazývam AND graf. Teraz počítač naloží rovnakým spôsobom z oboma ďalšími ťahmi. Pre druhý obdrží hodnotu -3 a pre tretí +1. Prvý ťah má najvyššiu hodnotu a preto ho počítač v partii použije. Tejto časti grafu, z ktorej vyberáme najvyššiu hodnotu, hovoríme OR graf.

Pri použití metódy minimax dochádza k zbytočnému vyšetrovaniu veľkej časti AND/OR grafu.

3.4 Alfa-beta

Zbytočnému prehľadávaniu AND/OR grafu sa dá zabrániť zavedením tzv. alfa a beta rezov. Alfa rezy zabránia zbytočnému vyšetrovaniu ťahov hráča A, beta rezy potom zbytočnému vyšetrovaniu ťahov hráča B. V skutočnosti je toto rozdelenie formálne a vyšetrovanie sa zastaví vždy, keď platí $\alpha \geq \beta$. Keď sa opäť pozrieme na obrázok 3.1, pri vyhodnocovaní počítač preskúma pozície (2) až (4) a pre svoj prvý ťah zistí hodnotu +2. Teraz začne skúmať druhý ťah (5) a pre koncovú pozíciu (6) nájde hodnotu -1. Tým už je isté, že druhý ťah by mal v najlepšom prípade hodnotu -1. Bude preto určite horší než ťah prvý. Počítač teda môže skúmanie druhého ťahu okamžite prerušiť a pozície (7) a (8) vôbec



Obrázok 3.2: Vyhľadávací strom s beta odsekávaním

neuvažovať. Na konečnom výsledku to nič nezmení. To isté platí aj pre skúmanie tretieho ťahu počítača. Hodnota +3 pre prvú koncovú pozíciu (10) spočiatku ospravedlňuje ďalšie hľadanie (keby boli všetky zostávajúce koncové pozície lepšie než +2, bol by tretí ťah ten najlepší). Akonáhle sa však objaví hodnota +1 pri ďalšej koncovej pozícii (11), je opäť zrejmé, že najlepší bol prvý ťah. Preto nie je nutné skúmať pozíciu (12).

Takéto skracovanie, pri ktorom sa neuvažujú možné ťahy protivníka, nazývame alfa odsekávanie. Obdobné skrátenie existuje aj u vlastných ťahov, ale samozrejme až po druhom polťahu. Uvažujme ako príklad časť vyhľadávajúceho stromu, ktorý dosahuje hĺbku troch polťahov (obrázok 3.2).

Program prevedie svoj prvý ťah, prvý ťah protivníka a teraz tri odvetné ťahy, ktoré má program k dispozícii. Očividne je treba oceniť (2) hodnotou +2 (z hľadiska programu), pretože sa pri najlepšej hre dá dosiahnuť pozícia (4) a tým aj táto hodnota. Teraz začneme so skúmaním druhého ťahu protivníka, ktorý vedie k pozícii (6). Program preverí prvý ťah a zistí pre pozíciu (7) hodnotu +4. Teraz už môže upustiť od prehľadávania (8) a (9) a to podľa nasledujúcej logiky: keď sa dostane do pozície (1), zistí môj súper pre pozíciu (2) hodnotu +2 (pre mňa). Pre pozíciu (6) vypočíta, že môžem dosiahnuť prinajmenšom +4. Preto radšej prejde k pozícii (2). Nemusí teda prepočítať hodnoty (8) a (9), pretože už (7) celú variantu vyvracia.

To isté platí v tretej vetve: V pozícii (11) to vyzerá tak, ako by protivník mohol nájsť lepšiu variantu. Program preto musí skúmať ďalej, aby videl, či je postupnosť ťahov ako celok pre protivníka lepšia. V pozícii (12) sa zistí, že je horšia než všetky možnosti za pozíciou (2). Pozícia (13) sa teda nemusí preverovať, pretože protivník sa bude snažiť prejsť k ťahu (2) než (10).

3.5 Iné algoritmy

Rôznymi mutáciami algoritmov minimax a alfa-beta môžeme tieto metódy vylepšiť bez narušenia ich presnosti, ale zvýšením rýchlosti ich ukončenia a nájdenia najlepšieho riešenia.

Môžeme rôznymi heuristickými metódami zmeniť poradie prehľadávania tak, aby algoritmus alfa-beta čo najskôr prestal prehľadávať ťahy, ktoré by aj tak odrezal. Napríklad ťahy, ktoré berú súperovu figúrku, by sa vyhodnotili skôr ako ťahy, ktoré neberú. Alebo ťahy, ktoré mali vysoké skóre v predchádzajúcich výmenách v analýze šachového stromu, by boli vyhodnotené prednostne.

Metóda nultého ťahu je metóda, ktorá preskakuje prehľadávanie v častiach stromu, kde sú pozície dostatočne dobré. Toto je overené vykonaním tzv. nultého ťahu (nevykonať ťah, akoby prenechať ťah súperovi) a prehľadávaním do redukovanej hĺbky. Ak výsledok tohto prehľadávania je väčší ako beta, nie je potrebné vykonávať ďalšie prehľadávanie. Ak je výsledok menší ako beta, je potrebné prehľadávať strom klasickou metóodou [4].

Kapitola 4

Evolučné algoritmy

4.1 Informatika inšpirovaná Darwinovou evolučnou teóriou

Rozvoj modernej informatiky je pozitívne poznamenaný skutočnosťou, že hľadá inšpiráciu v živej prírode, snaží sa formalizovať a implementovať paradigmy živej prírody pre návrh nových algoritmov, postupov a metód. Darwinova evolučná teória viedla ku vzniku evolučných algoritmov, ktoré patria v súčasnosti medzi aktuálnu problematiku informatiky a numerickej matematiky. Prekvapujúce na tejto novej netradičnej oblasti informatiky je, že okrem toho, že sa už stala organickou súčasťou modernej informatiky, stala sa tiež mostom medzi exaktnými prírodovedeckými oblasťami a vedami o živej prírode a určitou časťou humanitných vied.

Informatici majú teraz v rukách mocný simulačný nástroj, pomocou ktorého môžu získať relevantné odpovede na rôzne dôležité otázky a problémy evolučnej biológie, sociológie, psychológie atď., ktoré boli donedávna prístupné len špekulatívnym diskusiám založeným hlavne na analógiách alebo domnienkach autorov. Z pohľadu teórie a histórie vedy ide o unikátny fenomén modernej vedy. "Mosty" medzi rôznymi vednými oblasťami vznikajú obvykle ako výsledok intenzívnych interdisciplinárnych snáh preklenúť dve rôzne vedné oblasti. V prípade informatiky išlo o pomerne "jednostranný akt", aplikovali sa paradigmy živej prírody ako možné alternatívne algoritmy učenia a adaptácie a až neskôr sa ukázalo, aký silný simulačný prostriedok má informatika k dispozícii na pochopenie a interpretáciu javov a procesov prebiehajúcich v živej prírode. Na tejto situácii je pre informatikov fascinujúce to, že sa dostali do centra vedeckých aktivít, kde v rámci mierne modifikovaného argumentačného a pojmového aparátu informatiky získavajú relevantné výsledky týkajúce sa evolučných procesov v živej prírode alebo procesov prebiehajúcich na elementárnej úrovni neurónov v mozgu pri jeho kognitívnych aktivitách.

4.1.1 Prirodzený výber

Prirodzený výber, t.j. proces, v ktorom jedinci s veľkým fitness vstupujú do procesu reprodukcie s väčšou pravdepodobnosťou ako jedinci s malým fitness. Pod fitness rozumieme kvantitatívnu mieru schopnosti prežiť a vstupovať do reprodukčného procesu.

4.1.2 Náhodný genetický drift

Náhodný genetický drift, v ktorom náhodné udalosti v živote jedincov ovplyvňujú populáciu. Takýmito udalosťami sú napríklad náhodná mutácia genetického materiálu alebo náhodná

smrť jedinca s veľkým fitness predtým, ako mal možnosť zúčastniť sa reprodukčného procesu. Náhodné efekty genetického driftu sú významné hlavne pre malé populácie.

4.1.3 Reprodukčný proces

Reprodukčný proces, v rámci ktorého sa z rodičov vytvárajú potomkovia. Genetická informácia potomkov sa vytvára vzájomnou výmenou genetickej informácie rodičov. Najčastejšie tento proces prebieha tak, že z genetickej informácie dvoch jedincov sa náhodne vyberú časti chromozómu - informácie, z ktorých sa potom zostaví genetická informácia nového jedinca - potomka (tzv. sexuálna reprodukcia). Tento proces sa inak volá kríženie a vzhľadom na to, že sa vyskytuje u všetkých zložitejších organizmov, môžeme usúdiť, že podstatne zväčšuje rýchlosť a účinnosť evolúcie.

4.2 Prínos evolučných algoritmov do informatiky

Evolučné algoritmy založené na metafore Darwinovej evolučnej teórie predstavujú nový netradičný prístup k hľadaniu optimálneho (alebo suboptimálneho) riešenia zložitých optimalizačných problémov, ktoré nie sú riešiteľné klasickými technikami. Základným pojmom týchto algoritmov je populácia chromozómov, ktoré sú obvykle tvorené lineárnymi reťazcami symbolov. V týchto chromozómoch je zakódované aktuálne riešenie optimalizačného problému, chromozómy zodpovedajúce kvalitnému riešeniu sú ohodnotené veľkým fitness. Vstup do procesu reprodukcie je úmerný fitness chromozómov (chromozómy s veľkým fitness vstupujú s väčšou pravdepodobnosťou do reprodukcie). Táto reprodukcia obsahuje ešte aj určitý náhodný faktor (genetický drift) vyjadrený možnosťou náhodnej mutácie (v reťazci je náhodne vybraný symbol zamenený iným náhodne vybraným symbolom). Táto skutočnosť umožňuje evolúcii hľadať nové riešenia, ktoré sa v populácii ešte vôbec nevyskytli a môžu byť veľmi nádejné pre ďalšiu evolúciu populácie. Po určitom počte generácií sa začnú v populácii vyskytovať jedinci - chromozómy, ktoré reprezentujú vysokokvalitné riešenie daného optimalizačného problému.

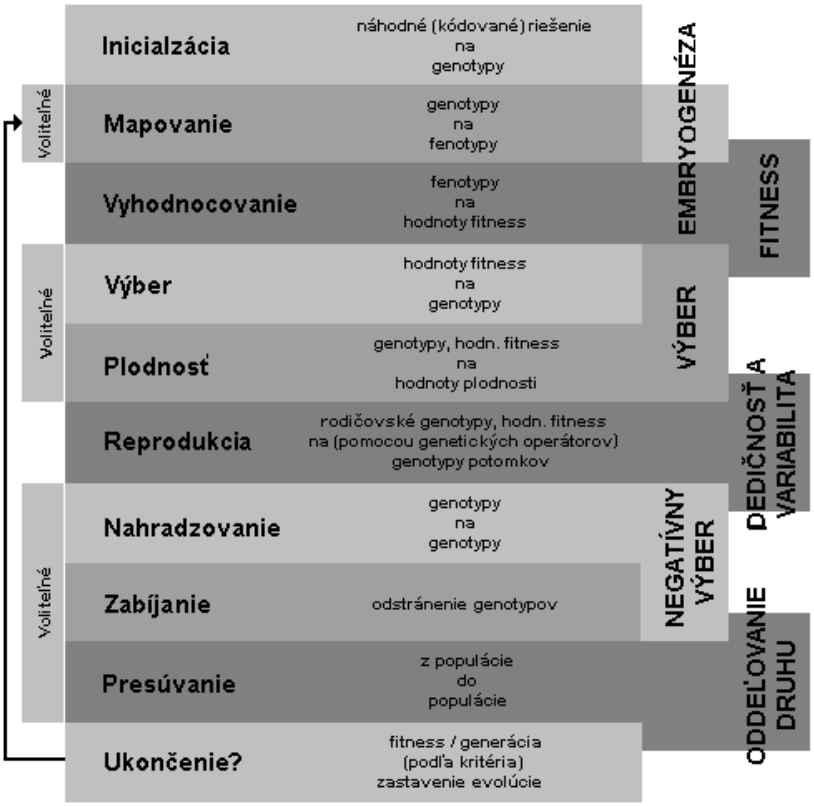
Vyššie popísaná základná idea evolučných algoritmov je založená na formalizácii Darwinovej evolučnej teórie, ktorá sa prevzala do informatiky z biológie. V súčasnosti už existuje v informatike celá paleta evolučných algoritmov, ktoré sú založené na rôznej interpretácii Darwinovej evolučnej teórie, na zdôraznení niektorého partikulárneho aspektu atď. Dokonca vznikli evolučné algoritmy, ktoré svoju inšpiráciu nehľadali v živej prírode, ale v neživej prírode - vo fyzike. Dobrou ilustráciou evolučných algoritmov založených na paradigme inej ako Darwinova evolučná teória je metóda simulovaného žíhania, ktorá vychádza z predstáv evolúcie termodynamických systémov.

Evolučné algoritmy v súčasnosti patria medzi základné nástroje modernej informatiky v prípadoch hľadania riešení v extrémne zložitých situáciách, kedy použitie štandartných deterministických metód založených na technikách úplného prehľadávania nemožno aplikovať (akoby sme hrali šach tak, že najprv zostrojíme všetky možné ťahy až do konca hry a z týchto vyberieme ten ťah, ktorý najrýchlejšie vedie k výhre). Ukazuje sa, že evolučná metafora je veľmi efektívnym prístupom k riešeniu týchto zložitých problémov najmä v prípadoch, keď nepotrebujeme optimálne riešenie problému, ale vystačíme aj s kvalitným suboptimálnym riešením. Základné vlastnosti týchto algoritmov môžeme zhrnúť takto:

1. Evolučné algoritmy patria medzi základné prostriedky modernej numerickej matema-

tiky pri riešení zložitých optimalizačných problémov. Používajú sa vtedy, ak hľadáme také globálne minimum, ktoré je obklopené množstvom lokálnych miním. Skutočnosť, že sa dajú takto použiť, je prekvapujúca (podobne ako pre neurónové siete vlastnosť, že sú univerzálnym aproximátorom funkcií), adaptáciou a modifikáciou všeobecných predstáv o Darwinovej evolučnej teórii sme dostali univerzálnu numerickú optimalizačnú metódu.

2. Evolučné algoritmy sa môžu vo všeobecnosti chápať ako abstrakcia a formalizácia Darwinovej evolučnej teórie. Pomocou evolučných algoritmov môžeme numericky simulovať darwinovskú evolúciu založenú na prirodzenom výbere a náhodnom genetickom drifte. Evolučné algoritmy poskytujú univerzálny algoritmus pre simuláciu evolúcie, v ktorom je potrebné modifikovať len spôsob určenia fitness chromozómu pomocou zodpovedajúceho reťazca symbolov (v biológii sa tento problém nazýva zobrazenie genotypu na fenotyp, kde lineárny reťazec chromozómu kóduje organizmus, ktorého schopnosť reprodukcie a prežitia je mierou fitness daného chromozómu). Táto skutočnosť otvára veľké možnosti pre informatikov zaoberajúcich sa genetickými algoritmami pre nové netradičné aplikácie v rámci evolučnej biológie, pri sledovaní evolučného vývoja tej-ktorej (aj sociálnej) vlastnosti, ktorá je zakódovaná v chromozóme [2].



Obrázok 4.1: Obecná štruktúra genetického algoritmu

4.3 Varianty evolučných algoritmov

Evolučné algoritmy majú niekoľko variant, najdôležitejšie sú štyri: Genetické algoritmy, genetické programovanie, evolučné stratégie a evolučné programovanie [5]. Všetky štyri boli vyvinuté nezávisle na sebe a najpoužívanejším z nich sú genetické algoritmy.

Obrázok 4.1 ilustruje priebeh evolúcie. Obrázok platí pre obecný evolučný algoritmus, ale niektoré fázy sú voliteľné a ich priebeh závisí na type algoritmu.

4.3.1 Genetické algoritmy

Genetické algoritmy patria medzi najpoužívanejšiu variantu evolučných algoritmov. Pri hľadaní riešenia problému sa využívajú reťazce číslic (tradične binárne, aj keď najlepším spôsobom reprezentácie je zvyčajne taký, ktorý odráža niečo o riešenom probléme), prakticky vždy sa aplikujú rekombinačné operátory spolu so selekciou a mutáciou. Tento typ evolučného algoritmu sa často využíva v optimalizačných problémoch.

Kapitola 5

Návrh evolučného algoritmu

Pri návrhu umelej inteligencie v hre šach pomocou evolučných algoritmov bolo potrebné sa zamerať na dve základné veci, t.j. na samotný evolučný algoritmus a na ohodnocovaciu funkciu. Ako evolučný algoritmus, tak aj ohodnocovacia funkcia sú veľmi dôležité súčasti programu a po generátore možných ťahov ide o, dá sa povedať, najdôležitejšie súčasti programu.

5.1 Chromozómy a gény

V každej populácii sa nachádzajú chromozómy (indivíduá, jedince). Skladajú sa z génov, ktoré sú teda základnými stavebnými jednotkami chromozómov. Gény nadobúdajú konečnú množinu hodnôt nad definovanou abecedou, sú nositeľmi informácie. V našom prípade bude gén reprezentovať jeden možný (legálny) ťah v postupnosti ťahov. Jednu postupnosť ťahov predstavuje jeden chromozóm.

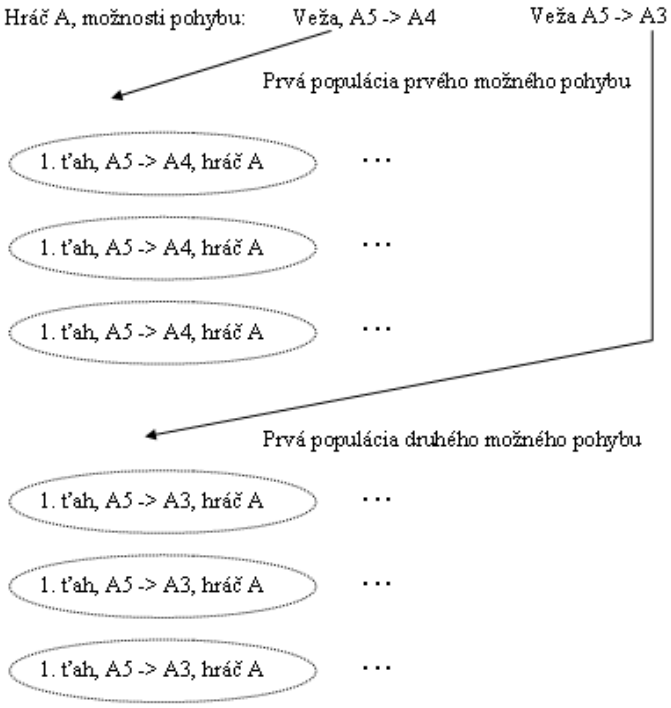
5.2 Navrhnutý evolučný algoritmus

Navrhnutý evolučný algoritmus pracuje s postupnosťami ťahov, ohodnocuje tieto postupnosti (chromozómy) a podľa číselnej hodnoty, ktorú vráti ohodnocovacia funkcia, rozhodne, ktorý chromozóm postúpi súťažou do procesu mutácie a vzniknuté chromozómy spolu s ich predkom tvoria novovytvorenú generáciu.

5.3 Etapy evolučného algoritmu

Pri prvom ťahu sa zoberú všetky figúrky hráča A, ktoré majú možnosť pohybu. Každá vybraná figúrka má teda na výber jeden alebo viac ťahov. Z každého možného ťahu každej figúrky sa vytvorí generácia o veľkosti niekoľkých chromozómov.

Po vygenerovaní počiatočnej generácie (kapitola 5.3.1) prebieha súťaž, výber najlepšieho jedinca v počiatočnej generácii (kapitola 5.3.2). Chromozóm s najvyšším fitness postúpi súťažou z predchádzajúcej generácie a tvorí základ novej generácie. Vybraný chromozóm je ponechaný nezmený a je súčasťou novej generácie. Ďalšie chromozómy vznikajú mutáciou vybraného chromozómu (kapitola 5.3.3). Týmto vznikne nová generácia. Teraz už opäť prebieha výber najlepšieho jedinca (kapitola 5.3.2), vytvorenie novej generácie (kapitola 5.3.3) atď. Toto sa opakuje, až kým nie je splnená podmienka vygenerovania vopred zadaného počtu generácií.



Obrázok 5.1: Spôsob vytvárania populácií

5.3.1 Počiatočná generácia

Prvým génom každého chromozómu generácie je informácia o prvom ťahu figúrky hráča A. Ďalšie gény chromozómov sa už generujú náhodne. T.j., ak je na ťahu napríklad hráč B, vygenerujú sa všetky možnosti jeho pohybu a náhodne sa jeden vyberie a doplní sa do chromozómu ako ďalší gén. Tento náhodný výber ťahov prebieha až do zadanej hĺbky, alebo kým nie je predchádzajúci ťak konečný (ukončenie môže byť spôsobené prehrou jedného z hráčov alebo ich remízou).

Ilustráciu vzniku chromozómov počiatočných generácií môžeme vidieť na obrázku 5.1. V načrtnutej situácii má hráč A iba dve možnosti pohybu, obe vežou z pozície A5. Prvým génom každého jedinca jednej populácie je jeden a ten istý ťah hráča A. Ďalšie gény chromozómov sa generujú náhodne a sú naznačené tromi bodkami za každým génom.

5.3.2 Výber najlepšieho jedinca

Po vygenerovaní populácie niekoľkých chromozómov prebieha výber (súťaž) potomkov. Toto je veľmi jednoduchý proces. Ide o porovnanie fitness hodnôt každého chromozómu jednej generácie a na ich základe výber jedného chromozómu s najvyšším fitness. Fitness chromozómu sa vypočíta sčítaním hodnôt všetkých génov. Hodnota génu je vypočítaná pomocou ohodnocovacej funkcie popísanej v kapitole 5.4.

5.3.3 Mutácia chromozómu

Jeden najlepší chromozóm, ktorý postúpi výberom z predchádzajúcej generácie sa klonuje a "vloží" do novej generácie. Mutáciou pôvodného chromozómu vznikajú ďalšie chromozómy

novej generácie, a to tak, že sa vyberie náhodný gén G_k z chromozómu predchádzajúcej generácie s poradovým číslom k od prvého génu chromozómu a ten sa vymení za gén (ťah) náhodne vybraného možného ťahu hráča, ktorý je v tejto postupnosti na pozícii k na rade. Pri mutácii nedochádza k zmene prvého génu chromozómu, teda $k \neq 1$.

Po výmene génov je potrebné zistiť, či sa za práve nahradeným génom nachádzajú nejaké ďalšie gény a ak áno, musí sa ešte skontrolovať, či tieto gény (v chromozóme s poradovým číslom vyšším ako k) vyhovujú novovzniknutej postupnosti. Pokiaľ toto neplatí, musia sa nevyhovujúce gény nahradiť náhodnými génmi, ktoré postupnosti vyhovujú.

5.4 Navrhnutá ohodnocovacia funkcia

Navrhnutá ohodnocovacia funkcia pracuje na pomerne jednoduchom princípe, t.j. v každom géne jedného chromozómu spočíta materiálovú hodnotu šachovnice - $\sum_{k=1}^{n}$ (materiálová hodnota figúrky k z celkového počtu n figúriek na šachovnici). Hodnoty figúriek sú použité štandartné, t.j. pešiak má hodnotu 1, strelec a kôň majú hodnotu 3, veža 5 a dáma 9. V prípade bielych figúriek sú tieto hodnoty kladné, v prípade čiernych záporné.

Po spočítaní materiálových hodnôt figúriek sa zistí, či náhodou hráč, ktorý potiahol, nedal súperovi šach mat. Ak sa tak stalo, vráti ohodnocovacia funkcia vysoké číslo (prípadne veľmi nízke, ak dal čierny hráč šach mat), ktoré je konštantné. Je to konštanta *1000000* resp. *-1000000*. Pokiaľ sa ťah, ktorým dal hráč šach mat v chromozóme nenachádza na prvom mieste (nejde o prvý ťah), delí sa táto konštanta pozíciou génu v chromozóme. V prípade remízy vracia ohodnocovacia funkcia hodnotu 0.

Kapitola 6

Implementácia

6.1 Java a Eclipse

Program bol implementovaný v programovacom jazyku Java 1.5 vo vývojovom prostredí Eclipse verzia 3.2.1. Napriek tomu, že interpretovaný jazyk Java môže byť oproti jazyku napríklad C/C++ pomalší, veľmi ľahko sa v tomto jazyku vyvíjajú aplikácie, je veľmi dobre zdokumentovaný, obsahuje veľké množstvo knižníc a je platformovo nezávislý.

Vývojové prostredie Eclipse používam už dlhšiu dobu, zvykol som si naň a som s ním spokojný. Eclipse poskytuje mnohé výhody oproti bežným textovým editorom alebo iným vývojovým prostrediam. Kontroluje syntax programu počas jeho písania, upozorňuje na možné chyby spôsobené programátorom, jednoducho sa vďaka jeho debuggeru hľadajú logické chyby v programe, za behu programu je možné vložiť do zdrojového kódu breakpoint(y) a začat s odlaďovaním programu a hľadaním chýb.

6.2 Popis implementovaných tried programu

V naprogramovanej aplikácii sa nachádza niekoľko tried. Z nich najdôležitejšie sú *Board, GeneticAlgorithm* a *MovesGenerator*. Funkcie niektoých z nich sú už podľa názvu zrejmé.

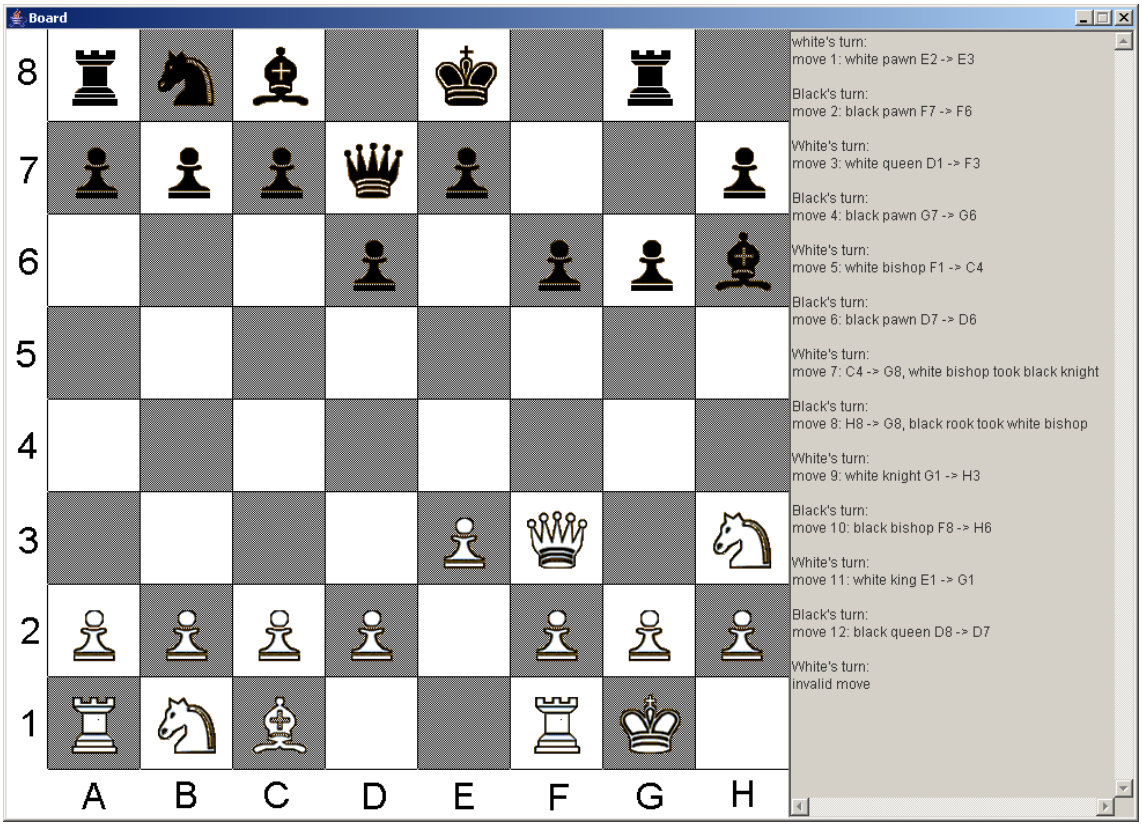
6.2.1 Trieda MovesGenerator

Hlavnou úlohou triedy *MovesGenerator* je generovanie možných ťahov hráča, ktorý je na ťahu podľa aktuálneho rozloženia figúriek na šachovnici. Obsahuje ďalej metódy, ktoré zistia napríklad, či je konkrétny ťah, zadaný parametrom pri volaní metódy, legálny (v súlade so šachovými pravidlami), alebo či nedostal jeden z hráčov šach mat.

6.2.2 Board

Pre vykresľovanie šachovnice a jej figúriek slúži trieda *Board*. Pre grafické užívateľské rozhranie som použil balík *javax.Swing*. *Board* implementuje rozhrania *MouseListener* a *MouseMotionListener* (z balíčka *java.awt.event*), reaguje teda na podnety užívateľa pracujúceho s myšou v rámci šachovnice. Pre skúmanie možných ťahov počítačom je potom následne potrebná aj metóda vrátenie sa o jeden krok na šachovnici späť. Pre tento účel je v triede *Board* implementovaná metóda *undo()*.

Trieda *Board* obsahuje objekt triedy *TextArea* z balíka *java.awt*. Sem sa zapisuje priebeh hry, ktorý je viditeľný užívateľovi. Nachádza sa v pravej časti okna so šachovnicou.



Obrázok 6.1: Pohľad na šachovnicu s textovým poľom zobrazujúcim priebeh hry

6.2.3 Trieda GeneticAlgorithm

V triede *Board* sa takisto nachádza informácia, ktorý hráč je na ťahu. Ak sa pohol užívateľ a na ťahu je počítač, vytvorí sa objekt triedy *GeneticAlgorithm*, ktorá obsahuje evolučný algoritmus, metódy mutácie, výberu apod. Po vytvorení objektu triedy *GeneticAlgorithm* sa zavolá metóda *Point[ ] evolve(boolean white)* vracajúca súradnice figúrky, ktorá sa má pohnúť a cieľové súradnice tohto pohybu.

Pseudokód metódy evolve:

```
public Point[] evolve(boolean white){
    data = vygeneruj všetky možné ťahy hráča určrného parametrom white;
    for (int x = 0; x < počet figúriek (určuje data); x++){
        for (int y = 0; y < počet ťahov figúrky x; y++){
            ancestors = vygeneruj prvú populáciu s prvým génom zadaným
                        parametrami x a y;
            for (int i = 0; i < počet generácií; i++){
                best = vyber najlepšieho predka z ancestors;
                chromosomes = pridaj best jedinca na index x;
                ancestors = vygeneruj novú generáciu, jeden z jedincov
                            bude chromozóm best;
```

```
            }
        }
    }
    chosen = vyber najlepšie chromozómy z chromosomes;
    vráť prvý gén (ťah) náhodne vybraného chromozómu z chosen;
}
```

6.2.4 Ďalšie triedy programu

Medzi ďalšie triedy programu patrí napríklad *Evaluation*. Tu sa nachádza statická ohodnocovacia funkcia vracajúca číselnú hodnotu materiálovej bilancie aktuálneho stavu šachovnice. Triedy *NewGameDialog* a *PromotionDialog* slúžia na zobrazenie dialógových okien, či už pri vytváraní novej hry v prípade *NewGameDialog* alebo zobrazenie dialógového okna s možnosťou výberu figúrky pri povýšení pešiaka po dosiahnutí ôsmej pozície v prípade *PromotionDialog*.

Kapitola 7

Dosiahnuté výsledky

V tejto kapitole by som chcel predstaviť dosiahnuté empirické výsledky v porovnaní s iným šachovými algoritmami.

Pre porovnanie uvádzam šachový program Chess nachádzajúci sa na internetovej adrese

`http://www.nabiscoworld.com/Games/game_large.aspx?gameid=10038`

využívajúci algoritmus minimax pre prehľadávanie stavového priestoru.

Tabuľka 7.1 predstavuje ťahy, kde môj navrhnutý algoritmus je biely a internetový šachový program čierny. Parametre evolučného algoritmu: počet génov v chromozóme je 6 (t.j. 6 polťahov), počet vygenerovaných generácií 200 a počet chromozómov v populácii 3. Program s navrhnutým evolučným algoritmom prehral. Jeden ťah, napríklad e2e4, označuje počiatočnú súradnicu figúrky (E2) a cieľovú (E4).

Poradie ťahu	**biely hráč**	**čierny hráč**	**poradie ťahu**	**biely hráč**	**čierny hráč**
1	e2e4	a7a5	10	g2g4	d6e5
2	b1a3	e7e5	11	f1d3	c8g4
3	h2h4	h7h5	12	f2f4	e5f4
4	d1h5	h8h5	13	h4h5	g7g6
5	d2d4	g8f6	14	h5g6	h7h1
6	d4e5	f6g8	15	g1e3	f7g6
7	a1b1	h5h7	16	e2g1	h1g1
8	e1d2	b7b6	17	b1a1	g1d1
9	c2c4	d7d6	18	d2c3	d8d3

Tabuľka 7.1: Prvá hra proti internetovému šachu

V tabuľke 7.2 sa nachádzajú ťahy z druhej hry proti tomuto internetovému šachovému programu. Opäť je tento čierny. Parametre evolučného algoritmu som zmenil nasledovne: počet generácií 300, počet chromozómov v populácii 5 a počet génov v chromozóme zostal nezmenený, t.j. 6. Dosiahnutý výsledok je o niečo lepší oproti predchádzajúcemu. Napriek tomu, že internetový šach mal materiálovú prevahu, výsledkom je remíza.

Výsledok po desiatich hrách vidieť v tabuľke 7.3.

Poradie ťahu	biely hráč	čierny hráč	poradie ťahu	biely hráč	čierny hráč
1	a2a4	e7e5	14	e2g1	a3c1
2	h2h4	d7d6	15	f1c4	b3c4
3	a1a3	h7h5	16	g2g4	c4d5
4	b2b4	f7f5	17	e1f2	d5h1
5	c2c4	d6d5	18	f2g3	h1g1
6	a3h3	d5c4	19	g3f3	g1g4
7	d1b3	c4b3	20	f3e3	g4a4
8	f2f4	d8d4	21	b5b6	a7b6
9	h3b3	d4f4	22	e3f2	a4h4
10	e2e4	f4e4	23	f2e3	a8a2
11	g1e2	e4c4	24	e3d3	a2d2
12	b1a3	c4b3	25	d3c3	c8e6
13	b4b5	f8a3			

Tabuľka 7.2: druhá hra proti internetovému šachu

	šachový program Chess	navrhnutý evolučný algoritmus
počet výhier	9	0
počet remíz	1	1

Tabuľka 7.3: celkový výsledok desiatich hier

7.1 Problém navrhnutého evolučného algoritmu

Navrhnutý algoritmus nevyberá ťahy veľmi "rozumne". Toto je spôsobené tým, že počas celej evolúcie sa zohľadňujú iba ťahy a materiálové hodnoty chromozómov hráča A. Teda akoby hráč A nepočítal s tým, že hráč B bude uvažovať pri reakciách na ťahy hráča A "rozumne" a že sa bude tiež snažiť vyhrať.

Kapitola 8

Iné spôsoby implementácie

V [7] zaujali evoluční návrhári celkom iný postoj k evolučnému riešeniu umelej inteligencie v šachu. Jeden z efektívnych spôsobov optimalizácie ohodnocovacej funkcie šachovej hry je prispôsobovanie každého z jej parametrov.

V prácach spojených s týmto problémom sa hodnoty týchto parametrov pohybujú v presne určenom rozsahu, čo by znamenalo, že akokoľvek sú mutačné a rekombinačné operátory aplikované, hodnoty týchto parametrov nemôžu presiahnuť tieto medze.

V práci [7] prišli návrhári s novým strategickým návrhom, ktorý nazvali stratégia dynamických hraníc. Teda hranice parametrov ohodnocovacej funkcie sú dynamické.

Ohodnocovacia funkcia počíta súčet materiálových hodnôt a) každej šachovej figúrky, b) počet dvojíc pešiakov nachádzajúcich sa v rovnakom stĺpci a c) počet možných legálnych ťahov (mobilita). Figúrkam sú prideľované rozličné váhy, ktoré sú neskôr prispôsobované učiacim sa procesom, aby bola optimalizovaná ohodnocovacia funkcia.

Pre prehľadávanie stavového priestoru je implementovaný algoritmus alfa-beta. Hĺbka prehľadávania sú 3 poľahy. Pre každú zajatú figúrku a pred každým pohybom figúrky sa vykoná rýchle preverenie stavu šachovnice. Toto poskytuje prehľadávanie ďalších 3 poľahov.

8.1 Optimalizácia ohodnocovacej funkcie evolučným algoritmom

Chromozóm reprezentuje reálny vektor veľkosti 6. Každý element vektoru (gén) reprezentuje váhu buď šachovej figúrky (okrem kráľa a pešiaka, ktoré sú hodnotené váhou pre kráľa 1000 a pre pešiaka 1), dvojitého pešiaka alebo mobility.

V každej populácii sa nachádza 5 chromozómov, ktoré medzi sebou súťažia, aby prežili. Dvaja jedinci (2 šachové programy používajúce 2 rôzne ohodnocovacie funkcie) hrajú proti sebe v dvoch hrách, kde v jednej je biely jeden jedinec a v druhej druhý jedinec. Ak individuum vyhrá, dostane jeden bod, inak dostane nula bodov. Na základe výsledkov z týchto dvoch hier je víťaz klonovaný a jeho klon mutovaný. Obe individuá, aj klon aj mutovaná verzia, sú skopírované do novej generácie. Tento proces pokračuje, až kým všetky jedince populácie prejdú týmto procesom.

Výber prebieha nasledovne. Najprv sa vyberie prvé individuum a to súťaží s náhodne vybraným jedincom z ostatných členov populácie. Víťaz je duplikovaný a porazený je nahradený týmto klonom. Následne je vybrané druhé individuum a to súperí s jedným

náhodne vybraným jedincom zvyšku populácie okrem prvého jedinca. Tento proces prebieha, až kým všetky jedince nie sú v súťaži zahrnuté.

Použitím tejto metódy výberu sa najlepší jedinec dostane na koniec vektoru populácie.

V tomto projekte je použitý real-coded mutačný operátor [6] ako hlavný operátor na vylepšenie parametrov ohodnocovacej funkcie každého jedinca. Jeho distribučný parameter η je nastavený na hodnotu 20. Mutovaný parameter y je nasledovný.

$$y_i^{(1,t+1)} = x_i^{(1,t+1)} + (x_i^{(U)} - x_i^{(L)})\vec{\delta_i}$$

Parameter $\vec{\delta_i}$ sa vypočíta z pravdepodobnosti:

$$P(\delta) = 0.5(\eta_m + 1)(1 - |\delta|^{\eta m})$$

$$\vec{\delta_i} = \begin{cases} (2r_i)^{1/(\eta_m+1)} & r_i < 0.5 \\ 1 - [2(1 - r_i)]^{1/(\eta_m+1)} & r_i \geq 0.5 \end{cases}$$

kde:

$y_i^{1,t+1}$ = nový parameter po mutácii
$x_i^{1,t+1}$ = parameter pred mutáciou
$x_i^{(U)}$ = parameter hornej hranice intervalu
$x_i^{(L)}$ = parameter spodnej hranice intervalu
η = pravdepodobnosť mutácie parametra v blízkosti originálneho parametra
r = náhodné reálne číslo na intervale [0 1].

Pravdepodobnosti mutácie pre dámu, vežu, strelca a koňa boli nastavené na 25%, zatiaľ čo pravdepodobnosti mutácie pre dvojitého pešiaka a mobilitu boli nastavené na 10%. Tieto hodnoty sú nastavené pomerne vysoko z dôvodu, aby došlo k mutácii aspoň jedného parametra a nenachádzali sa duplicitné chromozómy v jednej populácii.

Dobrá prehľadávacia metóda sa musí vysporiadať aj s veľkou rozmanitosťou prehľadávaného priestoru, takže nie je uväznená v lokálnom optime.

Pre vyriešenie problému konvergovania do lokálneho optima je využívaná stratégia dynamických hraníc. V iných prístupoch je potrebné uviesť spodnú a vrchnú hranicu pre každý parameter jedinca.

Základná myšlienka stratégie dynamických hraníc je mať prispôsobiteľné hranice pre každý parameter. Hranice každého parametra sú teda dynamické, aj keď rozsah je vždy rovnaký. Napríklad rozsahy pre dámu, vežu, strelca a koňa sú nastavené na 2. Rozsah pre dvojitého pešiaka je 0.2. Hranice mobility sú nastavené napevno (horná hranica = 0.5, spodná hranica = 0.01). Ak napríklad váha dámy je 9, potom jej vrchná hranica je 11 a spodná hranica 7. Ak by sa váha dámy znížila z 9 na 8.25, posunuli by sa teda hranice nasledovne: vrchná hranica by bola 10.25 a spodná 6.25. Ak váha dvojitého pešiaka je 0.5, jej hranice sú 0.7 a 0.3.

Pre urýchlenie učiaceho sa procesu váha dámy môže byť nastavená minimálne na váhu najvyššieho parametra jedincov (veže, strelca alebo koňa). Takto môžme uvažovať z dôvodu, že dáma by mala mať vždy vyššiu hodnotu ako hociktorá iná figúrka (mimo kráľa), pretože z hľadiska mobilnosti je najpoužiteľnejšia.

Najväčšia výhoda stratégie dynamických hraníc spočíva v tom, že prehľadávaný priestor v jednej generácii učenia je oveľa menší v porovnaní s prístupom pevne stanovených hraníc.

Kapitola 9

Záver

V predposlednej kapitole by som hlavne chcel zhrnúť a zhodnotiť výsledky mojich experimentov. Takisto sa chcem zamyslieť nad touto prácou z pohľadu jej významu a možností jej ďašieho vývoja, rozšírení.

Pri zhodnotení svojej práce musím povedať, že výsledok práce splnil svoje očakávania iba čiastočne. Poukazuje na to aj kapitola 7.1. Na navrhnutom algoritme by sa jeho nedostatky dali vyriešiť použitím stratégie typu: mutácia ťahov hráča A, výber najlepšieho chromozómu v prospech hráča A, mutácia ťahov hráča B, výber najlepšieho chromozómu v prospech hráča B. Výber najlepšieho ťahu by teda nebol zameraný len jedným smerom, zohľadňoval by potom aj inteligenciu hráča B. Týmto prístupom by som, podľa mňa, získal lepšie výsledky, ako boli dosiahnuté implementovaným algoritmom.

Vylepšiť by sa takisto dala aj ohodnocovacia funkcia, spočítanie materiálových hodnôt figúriek na šachovnici je veľmi málo, aj keď základný účel splní aj takáto ohodnocovacia funkcia. Do ohodnocovania by sa dala zahrnúť napríklad mobilita, teda počet možných ťahov hráča (čím viac možných ťahov, tým je ťah výhodnejší) alebo zistenie pozície dvojitého pešiaka (ak sa dvaja pešiaci nachádzajú v stĺpci za sebou, je to väčšinou považované za nevýhodu) apod.

Napriek tomu, že práca nenaplnila všetky očakávania, priučil som sa niečomu novému, obohatil som svoje vedomosti o oblasť informatiky, ktorú som veľmi predtým, než som na tejto práci začal pracovať, nepoznal.

Kapitola 10

Prílohy

10.1 Užívateľská príručka

Z pohľadu užívateľa sa celý program skladá z dvoch základných okien, je to okno s menu a okno so šachovnicou, kde sa zároveň nachádza aj textové pole pre textový popis priebehu hry. V menu sa nachádza iba niekoľko základných položiek, t.j. pre vytvorenie novej hry, pre ukončenie programu (obe vo *File*) a pre krok späť v rozohranej hre (nachádzajúce sa v záložke *Edit*). Po zapnutí aplikácie sa užívateľovi zobrazí iba okno s menu. Pokiaľ si želá vytvoriť novú hru, môže tak urobiť, ak zvolí *File* → *New game*. Zobrazí sa mu modálne okno s parametrami hry, môže si vybrať, proti komu bude hrať (hráč proti hráčovi alebo hráč proti počítaču). Po kliknutí na tlačítko OK sa užívateľovi zobrazí okno so šachovnicou so základným rozložením figúriek a s textovým poľom, zobrazujúcim priebeh hry.

Potom užívateľ, pokiaľ je na ťahu, môže pomocou tlačítiek myši ovládať svoje figúrky. Jednoducho iba klikne na figúrku, ktorou chce potiahnuť, pohne ňou na požadovanú pozíciu a stlačené tlačítko myši uvoľní. Ak je ťah v súlade so šachovými pravidlami, figúrka zostane na požadovanom mieste a vypíšu sa do textového poľa po pravej strane od šachovnice informácie skade, kam a aká figúrka sa pohla, prípadne akú figúrku vyhodila. Ak ťah nie je podľa šachových pravidiel legálny, figúrka sa presunie späť na miesto, skade vychádzala a do textového poľa sa vypíše informácia, že ťah nebol legálny. Príklad výpisov do textového poľa a grafické rozhranie šachovnice vidieť na obrázku 6.1.

Literatúra

[1] David B. Fogel. *Evolutionary Computation*. John Wiley & Sons, 2006. ISBN-13 978-0-471-66951-7.

[2] Vladimír Kvasnička, Jiří Pospíchal, and Peter Tiňo. *Evolučné algoritmy*. Vydavateľstvo STU v Bratislave, 2000. ISBN 80-227-1377-5.

[3] Dieter Steinwender and Frederic A. Friedel. *Šachy na PC*. UNIS Publishing, 1996. ISBN 3-87791-522-1.

[4] WWW stránky. Chess tree search. `http://www.cs.biu.ac.il/~davoudo/intro.html`.

[5] WWW stránky. Evolutionary algorithm. `http://en.wikipedia.org/wiki/Evolutionary_algorithm`.

[6] WWW stránky. Hybrid real-coded mutation for genetic algorithms applied to graph layouts. `http://www.cs.iusb.edu/~danav/papers/p323-vrajitoru.pdf`.

[7] WWW stránky. Using an evolutionary algorithm for the tuning of a chess evaluation function based on a dynamic boundary strategy. `http://www.cs.nott.ac.uk/~gxk/papers/ieeecis2006.pdf`.

[8] WWW stránky. Šachové myšlení. `http://www.linuxsoft.cz/article.php?id_article=1109`.