Modelovanie algoritmu pre šachový program*

Martin Čajka
Slovenská technická univerzita v Bratislave
Fakulta informatiky a informačných technológií
xcajkam@stuba.sk

14. december 2021

Abstrakt

Cieľom článku je objasniť štruktúru algoritmu šachového programu, vysvetliť myšlienkový postup za jej jednotlivými časťami a zanalyzovať univerzálne používané prístupy tvorenia algoritmu pri takej komplexnej hre, akou je šach. Článok sa taktiež zameriava na riešenie problémov a otázok s ktorými sa osoby podieľajúce na vývoji algoritmu stretnú a popisuje metódy ich riešenia, ako napríklad využívanie prvkov umelej inteligencie alebo ponáranie sa do princípov strojového učenia. K záveru sa článok vyjadrí k fungovaniu algoritmu AlphaZero, procese jeho tréningu a následného porovnania s vtedajšími najsilnejšími programami pre jednotlivé hry.

Kľúčové slová: šachový algoritmus, neurónové siete, strojové učenie, umelá inteligencia a šach

1 Úvod

„Šach je všetkým: umením, vedou aj športom“ – vyhlásil ruský šachový veľmajster Anatolij Karpov. Za posledné dve dekády sa vďaka rapídnemu pokroku v oblastí umelej inteligencie tento jeho výrok aspoň z jednej časti potvrdil – šachu vládnu stroje, ktoré dominujú ľudských protivníkov na základe ich bleskových kalkulácii a schopnosti naozaj sa poučiť z vlastných chýb. Je fascinujúce ako rýchlo prebehla evolúcia v oblasti šachových strojov. Od čias šachových majstrov skrývajúcich sa v "Mechanickom Turkovi"to už je síce 250 rokov, prvé softvéry schopné naozaj hrať šach sa začali vyvíjať pred približne 50 rokmi.

2 Základy a materiálová hodnota

Aj napriek tomu, že šach má jasne stanované pravidlá a určité limitácie pre pohyb figúrok, stále tu je veľmi vysoká variabilita - každý jeden ťah má značný dopad na štádium hry. Existujú však isté faktory, ktoré napovedajú ako je na

*Semestrálny projekt v predmete Metódy inžinierskej práce, ak. rok 2021/22, vedenie: Vladimír Mlynarovič

tom hra a ktorý hráč ma najlepšie predpoklady zvíťaziť. Jedným z najznačnejších je materiálová hodnota figúrok. Jednotlivé figúrky majú všeobecne známu priradenú hodnotu - napr. jazdec a strelec sú obidvaja známy pod hodnotou 3 body, no nie je to také jednoduché. Ich hodnota taktiež závisí od viacerých faktorov - od pozície, teda ako dobre pokrývajú ostatné figúrky a zároveň v akej miere ohrozujú súperove figúrky a možné ťahy, ako aj od štádia hry, napríklad ku koncu hry kde je menej figúrok je dvojica strelcov nadradená dvojici jazdcov, nakoľko majú väčšiu mobilitu ako aj synergiu, keďže sú schopné voľnejšie sa hýbať po šachovnici a pokryť dve celé diagonály. Otázkou je teda ako nastaviť algoritmus tak, aby bol schopný robiť ťahy ktoré ho dovedú k lepšej pozícií a v konečnom dôsledku aj ku výhre?

2.1 Reprezentácia plochy a pohybu

Prirodzene, základné šachové pravidlá a pohyby sú do šachového programu vštiepené na začiatku jeho vývoja. Ďalším krokom je potrebné však jasne definovať aké je v ktoromkoľvek momente rozloženie figúr, aké majú možné ťahy a ako sa navzájom ohrozujú alebo zakrývajú.

Reprezentovať stav figúr sa dá viacerými spôsobmi. Najintuitívnejším a aj často používaním riešením je použiť jednorozmerné pole alebo dvojrozmerné polia. V dvojrozmernom poli sa dá indexovať prvky kombináciou riadku a stĺpca, nevýhodou však býva efektivita, nakoľko takýto program potrebuje viac kontrolných podmienok aby nedošlo k chybe v pamäti. V jednorozmernom poli sa políčka indexujú postupne a šetrí sa aritmetickými funkciami v porovnaní s dvojrozmerným poľom. Iné často používané reprezentácie šachovnice sú napríklad metóda 0x88 a bitboardy. V metóde 0x88 sa používa pole veľkosti 8x16, teda dve veľa seba sediace šachovnice, jedna plocha je pre šachovnicu a druhá na kontrolu správnosti ťahov. Pri použití bitboardov sa využíva 64-bitová sekvencia 0 a 1, ktorá hovorí či existuje na danom políčku nejaká figúrka. Stav hry sa môže reprezentovať pomocou viacerých bitboardov, jeden pre každý typ figúrky. [?].

Pre účel pohybu figúrok sa vo väčšine prípadov algoritmus modeluje nasledovne: Každé políčko na šachovnici reprezentuje vektor s dĺžkou 64 tak, že každý komponent vo vektore smeruje k možnej pozícii na šachovnici. Tieto komponenty neskôr môžu nabrať rôzne hodnoty ktoré budú reprezentovať buď figúrky daného hráča alebo prázdne polia [1]. Teraz ale už nastáva čas, aby program začal robiť svoje prvé vlastné ťahy.

3 Otvorenia

Pri začiatočnej fáze hry je potrebné algoritmus nejakým spôsobom usmerniť ako má postupovať. Univerzálne sa aplikujú dva postupy. Menej sofistikované algoritmy sa držia nejakým všeobecným pravidlám ako rozvíjať svoje jednotlivé figúrky. Tieto pravidlá sa dajú vizualizovať ako mapy šachovnice pre, ktoré priťahujú figúrky do určitých zón. Pre kráľa je napríklad esenciálne aby so dostal do bezpečia, teda jeho sa odporúča aby smeroval do rohov šachovnice. Naopak zas figúrka jazdca má svoju komfortnú zónu bližšie k strede šachovnice, kde má k dispozícii viac možných ťahov ako na kraji. Práve kvôli tomuto princípu je potrebné cielene formovať pomocou pravidiel program tak, aby usmerňoval svoje

otváracie štádium hry na to aby posilnil svoju kontrolu nad centrom šachovnice. Druhý postup ktorý už využívajú pokročilejšie šachové programy sa zakladá na využití princípoch strojového učenia. Pre algoritmus sa stiahne rozsiahla databáza najhrávenejších a najúspešnejších otvorení ako aj otvorenia odohraté na najvyššej úrovni. Z tejto databázi algoritmus čerpá a modifikuje svoje otvorenia, ale pri následnom vývine hry už prichádzajú do vyhodnocovania pozície komplexnejšie funkcie.

4 Vyhodnocovanie a vyhľadávanie ťahov

Logický obsah posledných dvoch kapitol nám dáva pohľad na faktory, ktoré sa podieľajú na generácii ťahov algoritmu. Generácia ťahov sa skladá z dvoch častí: vyhodnocovacej a vyhľadávacej funkcie. Vyhodnocovacia funkcia hodnotí stav hry z relatívného ohodnotenia pravdepodobnosti výhry istého hráča, zatiaľ čo vyhladávacia funkcia je predbežný pohľad do možných štádii hry použitím minimax algoritmu. [3]. Minimax algoritmus je isté rozhodovacie pravidlo v teóri hier, ktorého podstatou je minimalizovať potenciálne "straty" hráča a zároveň maximalizovať zisk skontrolovaným všetkých možných situácii ktoré môžu nastať. V šachu ide o získanie výhody vyhodnotením všetkých možných ťahov a následnych možných reakcí protihráča. Daný algoritmus je schopný zájsť do určitých hĺbok, teda pozrieť sa na kombinácie ťahov jedného a druhého hráča niekoľko ťahov dopredu. Šachové algoritmy reprezentujú situáciu hry hodnotou, ktorá môže naberať kladné a záporné hodnoty. Kladné hodnoty znamená že hráč ovládajúci biele figúrky má situačnú výhodu, teda sa očakáva že má väčšiu pravcdepodobnosť zvíťaziť, zatiaľ čo záporné hodnoty poukazujú na výhodu hráča s čiernimi figúrkami. Napr. hodnota -5.1 znamená že čierny hráč vedie o daný počet bodov, tedá má buď materiálnu alebo aj situačnú výhodu.

V praxi sa tieto funkcie realizujú pomocou umelých neurónových sietí. Podobne ako neuróny v ľudskom mozgu, umelé neurónové sieťe sú namodelované ako jednotlivé uzly reprezentujúce neuróny, ktoré sú navzájom pospájané. Skupiny neurónov sa nachádzajú na môžu nachádzať na viacerých úrovniach alebo vrstvách, záležiac od zložitosti algoritmu ktorý sa tvorí. Jednotlivé neuróny príjmajú vstupy, zvažujú situáciu a následne dávajú výstup. Umelé neurónové siete sa požívajú na riešenie problémov umelej inteligencie ktorá pomocou ních rozmýšla, nachádza vzory, vzťahy medzi vstupom a výstupom a vyhodnocuje situáciu. V prostredí šachu sa spája hlavne so strojovím učením, ktoré tvorí svoje dáta a referencie tak, ako to bolo popísané v predchádzajúcom odstavci.

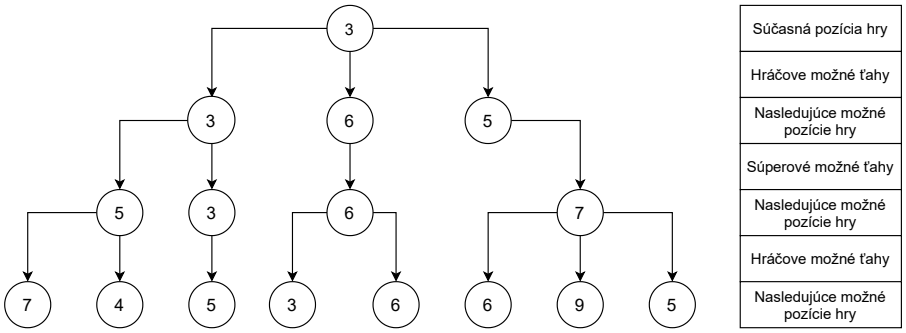
4.1 Prístupy

Existujú rôzne spôsoby podľa ktorých sa dá generácia ťahov algoritmu upraviť. Cieľom tejto úpravy je algoritmus urobiť efektívnejší, napríklad zmenou správania vyhodnocovacej alebo vyhľadávacej funkcie. Najčastejšie používané sú bližšie opísané v nasledujúcom zozname.

- Quiescence Search Quiescence search sa dá voľne preložiť ako "vyhľadávanie tichej pozície". Zmena v správaní algoritmu nastáva pri vyhodnocovacej funkcii, ktorá odkladá evaluáciu pri pozíciách ktoré nie sú "pokojné" a vyhľadáva ďalšie ťahy kým sa nedostane do pozície v ktorej neexis-

tujú žiadne hrozby. Funkcia teda neprehľadáva pozície do nejakej fixovanej hĺbky, ale dovtedy, kým je to nutné.

- Alpha-Beta Prunning Alpha-Beta prunning je vyhľadávací algoritmus, ktorý zefektívňuje vyhľadávanie tým že znižuje počet vetiev ktoré prehľadáva. Funguje na spôsobe ohodnotenia každej vety nejakou číselnou hodnotou. Pokiaľ je hodnota výrazne horšia - kritéria porovnávania môžu byť rôzne, napríklad nižšia/vyššia hodnota vetvy - algoritmus danú vetvu odpíše a už ju ďalej neprehľadáva.



Obr. 1: Model Alpha-Beta Prunning algoritmu - (preložené a upravené) [2]

- Monte Carlo Tree Search Algoritmus Monte Carlo Tree Search funguje nasledovným spôsobom. Vyhľadávacia funkcia hľadá vetvy so sľubným rozvojom hry, dané pozície odohrá do konečného výsledku všetkými možnými sekvenciami ťahov a priradí vetve hodnotu finálnych výsledkov, napríklad v tvare podielu - výhry/remízy/prehry - a porovná vetvy. Následne pokračuje vetvou, ktorej množina výsledkov je najperspektívnejšia.

Väčšina algoritmov obsahuje vo svojej štruktúre isté nastavanie alebo funkciu, ktorá ho rozlišuje od ostatných. Aj v oblasti kompetitívnych šachových algoritmov sú stále rozličné prístupy pri budovaní najlepších z najlepších. Vyzerá to tak, že neexistuje metóda, ktorá by sa ukázala ako nadradená, ale je potrebné poznamenať že algoritmy sú hodnotené podľa výsledkov súťaže v turnaji so špecifickými podmienkami.

5 AlphaZero

AlphaZero je počítačový program vytvorený na pochopenie a dosiahnutie vysokej úrovne hrania v doskových strategických hrách pre dvoch, konkrétne šach, shogi a go. Ide o hry ktoré sú známe svojou komplexnosťou a vysokou variabilitou. Majú určité pravidlá no stratégia hry neskôr záleží od samotného hráča. AlphaZero bol smerovaný rovnako ako obyčajný hráč ktorý sa rozhodol hrať dané hry. Na rozdiel od ostatných algoritmov AlphaZero bol naučený len

základné pravidlá hry a princíp jeho učenia bol postavený na základe reinforcement learningu, teda metódy strojového učenia ktorá odmeňuje alebo dáva pozitívnu spätnú väzbu pri ťahoch ktoré zlepšujú situáciu hráča a naopak, negatívnu spätnú väzbu pri ťahoch zhoršujúcich hráčovo postavenie [?]. Druhá metódy skombinovaná s reinforcement learning bola metóda "self-play", algoritmus hral partie sám so sebou, robil ťahy a analyzoval situáciu pre obidve strany. Namiesto vyhodnocovacej funkcie AlphaZero využíval neurónovú sieť so vstupnou hodnotou reprezentujúci pozíciu hry, výstupným vektorom s možnými ťahmi a skalárnej veličiny predpovedajúcej očakávaný výsledok daného ťahu. Prirodzene počtom odohratých hier sa neurónová sieť zlepšovala a odohraté hry ju smerovali pri vyhľadávaní ťahov.

Na tréning algoritmu bolo stanovených 24 hodín, za ktorých bolo cieľom dosiahnúť a prekonať úroveň vtedajších najsilnejších programov v jednotlivých hrách. Pre účely tréningu sa použili špeciálne TPU integrované obvody od firmy Google vyvinuté na strojové učenia pomocou neurónových sietí. Algoritmus AlphaZero používal 5000 TPU obvodov prvej generácie na tvorenie hier metódou self-play a 64 TPU obvodov druhej generácie na tréning neurónových sietí. Už po štyroch hodinách tréningu tím pracujúci na algoritme predpovedal že AlphaZero dosiahol úroveň Stockfishu 8, algoritmus ktorý bol doposiaľ hodnotený ako najsilnejší. Tréning bol zavŕšený časovaným turnajom pozostávajúceho zo 100 hier medzi AlphaZero a jednotlivými programami. Počas turnaja bežal algoritmus len na 4 TPU obvodoch a 44-jadrovom procesore, čo je štandardný hardvér pri turnajových podmienkach. [?]

	Šach	Shogi	Go
Trénovací čas	9h	12h	34h
Počet hier	44 mil.	24 mil.	21 mil.
Čas na ťah	40 ms	80 ms	200 ms

Obr. 2: Štatistiky tréningu AlphaZero (upravené a preložené) - *čas tréningu pre šachový algoritmus trval len 9 hodín, za daný čas bol algoritmus schopný odohrať 44 miliónov hier. Hornú hranicu jedného ťahu predstavovalo 40 milisekúnd.* [?]

Hra	Biely	Čierny	Výhra	Remíza	Prehra
Šach	AlphaZero	Stockfish	25	25	0
	Stockfish	AlphaZero	3	47	0
Shogi	AlphaZero	Elmo	43	2	5
	Elmo	AlphaZero	47	0	3
Go	AlphaZero	AG0 3-day	31	-	19
	AG0 3-day	AlphaZero	29	-	21

Obr. 3: Výsledky porovnávania AlphaZero s najlepšími programami v daných hrách (preložené) - *V turnaji pozostávajúceho zo 100 hier šachový algoritmus AlphaZero porazil Stockfish výsledkom 28/72/0 (výhry/remízy/prehry).* [?]

6 Reakcia na témy z prednášok

Prezentácia: slajdy a prednes Ukážkou vedomosti nie je používať odborné slová pre danú oblasť, ale umenie vedieť vysvetliť daný koncept aj dieťaťu. Schopnosť správne zdieľať a pochopiť informáciu je v dnešnom svete veľmi dôležitá, no spoločnosť s ňou má stály problém. Z každej strany sme bombardovaný množstvom informácii a náš mozog je veľkú množinu času prestimulovaný a nie je schopný všetky tieto informácie spracovať. Jedna z foriem zdieľania informácii s ktorou sa často stretávame je prezentácia a hoci sme s ňou boli v kontakte od skorých školských čias, je to forma na ktorú mnoho ľudí neberie veľký ohľad a podceňuje ju.

Ja osobne si myslím, že základom správnej prezentácie je aby bola pamätihodná, aby spravila dojem na publikum. Keď je človek vystavený niečomu, čo ho zaujíma, je unikátne alebo mimo jeho očakávaní, jeho vnemy zbystrú a schopnosť nasávať informácie sa výrazne zlepší. Spraviť pamätihodnú prezentáciu vieme pomocou správneho použitia prvkov prezentácie. Dôležité je aby slajdy boli ľahko zrozumiteľné, nebolo na nich veľa textu, použitie kvalitných obrázkov, využitie relevantných grafov a správne zladiť farby prezentácie s témou. Taktiež dôležitá časť prezentácie je aj prednes. Chceme aby nás publikum počúvalo, aby bolo angažované. Je potrebné zvoliť zodpovedajúcu reč tela myšlienke ktorú sa snažíme vyjadriť, takmer vždy je dobrým základom úsmev a očný kontakt. Náš hlas je tiež členom rovnice pre dobrú prezentáciu. Treba vedieť sa pohrať z tónom hlasu, dynamikou a hlavne najlepšie je, keď náš hlas znie živo, že toto je téma, o ktorú sa naozaj zaujímame.

Inžinierska gramotnosť a informatika V oblasti informatiky je inžinierska gramotnosť podľa môjho názoru veľmi relevantným pojmom. Každý, kto so rozhodne venovať informatike, musí byť vyzbrojený značnou dávkou vynaliezavosti, disciplíny a kompetencie. Či už sa jedná o riešenie úloh pri programovaní alebo pri tvorbe prelomového softvéru, sú nevyhnutné isté vlastnosti, ktoré spolu tvoria inžiniersku gramotnosť.

V informatike sa počítače stávajú pravou rukou inžiniera, nakoľko jedným z hlavných zdrojov na ktorých inžinier v informatike buduje, sú dáta. Pomocou programov alebo algoritmov prichádza inžinier k novým poznatkom alebo nachádza súvislosti ktoré nie sú voľným okom jasné. Prirodzene, inžinier musí nájsť spôsob alebo riešenie ako takéto programy a algoritmy zostrojiť, tu už je potrebné aby využil svoje vlastné zručnosti a priniesol na svet myšlienku. Získane informácie by následne mal byť inžinier schopný správne interpretovať aby na nich mohol budovať a posúvať svoje myslenie vpred.

Bibliografia a citovanie v technickom texte Bibliografia a citácie sú perfektným spôsob ako použiť a oceniť prácu iných a nedopustiť sa plagiátorstva. Bibliografia sa väčšinou uvádza na konci práce a obsahuje dôležité informácie ako napríklad názov diela z ktorého čerpáme, meno autora, vydavateľa, rok vydania a podobne. Niekedy sa uvádza aj krátky popis čomu sa daná literatúra venuje. Bibliografiu sa v texte referencuje pomocou čísiel alebo mena autora a dátum pri častiach alebo vetách ktoré boli použité v našej práci. Citácia môže mať tvar priamej citácie, parafrázovania alebo zovšeobecnenie práce autora. V technickom texte bibliografia zároveň slúži aj ako odkaz pre čitateľa, ak by sa

chcel o problematike dozvedieť viac.

Občas môže byť problémom nájsť bibliografické informácie. Mne osobne sa stalo, že som mal problém nájsť bibliografiu v akomkoľvek tvare, už som chcel uviesť citáciu bez bibliografie ale uznal som že by to nebolo ideálne, nakoniec som sa dokázal vynájsť a spraviť bibliografiu z dostupných údajov o práci, ktoré som na internete mohol nájsť.

7 Zhodnotenie

Na záver by chcel článok poskytnúť stručný prehľad a zhrnúť svoj obsah. Článok postupne prechádza štruktúrou a podrobnejšie sa venuje jej jednotlivým častiam. K začiatku sa vyjadruje k všeobecnej šachovej logike, ktorú je potrebné ovládať pri tvorbe šachového algoritmu. Následne sa venuje metódam reprezentácie plochy šachovnice, ich výhodám a nevýhodám a taktiež aj reprezentácií pohybu pomocou vektorov. V tretej kapitole článok opisuje myšlienkový postup tvorenia silného jadra v otváracej časti hry alebo použitie strojového učenia či databáz. Po otvoreniach prichádza na rad najdôležitejšia časť šachového algoritmu, ktorou je princíp generácie ťahov. Vo väčšine prípadov sa skladá z vyhodnocovacej a vyhľadávacej funkcie, ktoré pomocou neurónových sietí vyhodnocujú postavenie hry a navrhujú ďalší ťah. V tejto časti článok analyzuje aj rozdielne prístupy ktoré sa dajú zvoliť pri tvorbe tejto časti algoritmu, ako napríklad Alpha-Beta Prunning alebo Monte Carlo Tree Search. Posledná riadna kapitola sa venuje konkrétnemu šachovému algoritmu AlphaZero a princípe jeho tvorby na základoch strojového učenia. Pri jednotlivých častiach je úlohou článku priblížiť inžinierske myslenie ktoré je nevyhnutné pri tvorbe algoritmu ako aj riešení naskytujúcich sa problémov.

Literatúra

[1] D.B. Fogel, T.J. Hays, S.L. Hahn, and J. Quon. A self-learning evolutionary chess program. *Proceedings of the IEEE*, 92(12):1947–1954, Dec 2004.

[2] S.H. Fuller, J.G. Gaschnig, J.J. Gillogly, CARNEGIE-MELLON UNIV PITTSBURGH PA Dept. of COMPUTER SCIENCE., and Carnegie Mellon University. Computer Science Department. *Analysis of the Alpha-beta Pruning Algorithm*. Carnegie-Mellon University. Department of Computer Science, 1973.

[3] Barak Oshri. Predicting moves in chess using convolutional neural networks. 2015.