MINUT 2023 (5), s. 195-210
ISSN 2719-3063

Piotr SŁANINA

Katedra Matematyki, Politechnika Śląska, ul. Kaszubska 23, 44-100 Gliwice

Co nieco o algorytmach szachowych

Streszczenie. Algorytmy szachowe i ich implementacja od wielu lat są istotną częścią szachowego świata. Artykuł przybliża historię i działanie podstawowych algorytmów, wykorzystywanych w silnikach szachowych.
Słowa kluczowe: algorytm szachowy, twierdzenie o minimaksie, alfa-beta, metoda Monte Carlo, Stockfish, AlphaZero

1. Wstęp

Szachy w pierwotnej formie powstały w Azji około 1500 lat temu i z czasem stały się jedną z najpopularniejszych gier planszowych. Początkowo były traktowane jako rozrywka, jednak najlepsi szachiści mogli liczyć na coraz większe korzyści. Najwyższa dotychczasowa pula nagród w szachowym pojedynku to 5 milionów dolarów w towarzyskim meczu pomiędzy Robertem Fischerem a Borysem Spasskim w 1992 roku. Obecnie wiele osób utrzymuje się dzięki szachom. Najlepsi gracze mogą liczyć na przyzwoite nagrody. Zarobić można również na trenowaniu i prowadzeniu zajęć szachowych w szkołach i przedszkolach. W ostatnich latach pojawiło się sporo zawodowych streamerów szachowych. Według serwisu Alexa, w 2022 roku chess.com był 194 stroną na świecie pod względem ruchu sieciowego i zatrudniał ponad 400 osób.

Odpowiedzią na zainteresowanie szachami była systematyzacja dotychczasowej wiedzy i rozwój teorii szachów. Do dzisiaj powstało wiele opasłych książek, poświęconych debiutom (początkowa faza gry), niezliczone zbiory zadań czy dogłębne opracowania, dotyczące gry końcowej.

Przełomem było zaprzęgnięcie komputerów do pomocy szachistom. Algorytmy, używane w programach grających w szachy (zwane też silnikami szachowymi), były coraz doskonalsze i przystawały do aktualnych możliwości sprzętowych. Obecnie nawet szachowi amatorzy posiłkują się programami szachowymi w przeglądzie partii.

Już ponad 100 lat temu powstał pierwszy automat, grający proste końcówki wieżowe. Następnie pojawiły się pierwsze programy szachowe. Do wyszukiwania najlepszych posunięć wykorzystywały głównie algorytm min-maks, jego ulepszenie alfa-beta i kolejne, bardziej zaawansowane wersje. Później pojawianie się coraz szybszych komputerów ułatwiało przekraczanie kolejnych granic w pojedynkach człowieka z maszyną. O ile jeszcze w latach siedemdziesiątych poziom gry programów szachowych wywoływał uśmiech politowania na twarzach szachistów, to już w 1989 roku Bent Larsen przeszedł do historii jako pierwszy

Autor korespondencyjny: P. Słanina (piotr.slanina@polsl.pl).
Data wpłynięcia: 23.08.2023.

arcymistrz, który przegrał w szachy z komputerem (program Deep Thought). Ulepszona wersja tego programu, Deep Blue, w 1996 roku wygrała z aktualnym Mistrzem Świata, Garri Kasparowem. W XXI wieku w walce z czołowymi silnikami szachowymi, jak Stockfish, Houdini czy Komodo, nawet czołowi gracze świata stoją na straconej pozycji. W sierpniu 2023 roku, stosowany w szachach ranking ELO programu Stockfish 15 wynosił ponad 3500, a ranking najlepszego szachisty świata, Magnusa Carlsena był równy 2835. Czy to duża czy mała różnica? Otóż jeżeli a jest rankingiem ELO gracza pierwszego, a b – drugiego, to wartość oczekiwana liczby punktów gracza b uzyskanych w partii, rozegranej pomiędzy graczami a i b, wynosi

$$W(a,b) = \left(1 + 10^{\frac{a-b}{400}}\right)^{-1}.$$

Stąd można wyliczyć, że na 100 rozegranych partii, oczekiwana liczba punktów, zdobyta przez Stockfish 15 w starciu z Carlsenem, wynosi niemal 98!

Rewolucyjną koncepcją w podejściu do tworzenia programów, grających w gry turowe, okazało się wykorzystanie sieci neuronowych do implementacji algorytmów samouczących się. O wadze tego osiągnięcia świadczy uznanie go przez czasopismo Science za jeden z przełomów roku 2016. Na tej koncepcji bazuje program AlphaGo (do gry w grę Go), a także późniejszy AlphaZero (stworzony do gry w dowolne gry turowe, między innymi szachy). AlphaZero bez problemu rozprawiał się z dotychczasowymi liderami list najsilniejszych programów szachowych.

Twórcy innych czołowych programów szachowych nie pozostali dłużni, a ich najnowsze wersje również zaczynają korzystać z sieci nieuronowych i udoskonalonego samouczenia się. Bazujący na AlphaZero program AlphaZe** jest najnowszym osiągnięciem w dziedzinie programów, grających w gry [2]. Wyszukuje on najkorzystniejsze zagrania w grach, niekoniecznie będących grami turowymi – czyli w grach, w których gracze nie znają wszystkich decyzji rywali w chwili podejmowania swoich lub pojawiają się elementy losu.

W trakcie opisywania matematycznych pojęć, uprościmy nieco język z korzyścią dla czytelników mniej biegłych w teorii gier i algorytmów. Między innymi, w teorii gier istnieje formalna definicja *gry*, ale nie musimy jej tu przytaczać. Ustalmy jedynie, że w każdej *grze* wszyscy gracze muszą znać jej zasady, a po zakończeniu *gry* każdy z graczy otrzymuje w nagrodę pewną liczbę punktów, będącą liczbą rzeczywistą. Celem każdego z graczy jest uzyskanie jak największej liczby punktów. Jeżeli reguły *gry* nie określają inaczej, to zakładamy, że zwycięstwu w *grze* odpowiada wartość 1, remisowi 0, a przegranej -1.

Rodzina *gier* jest bardzo pojemna i zawiera dość dziwne gry. Przykładowo, w grze, którą nazwiemy nazwiemy *zachłanni*, zasady określimy następująco: n graczy wybiera jednocześnie po jednej liczbie rzeczywistej. Każdy z graczy zdobywa liczbę punków równą wybranej przez niego liczbie.

Zauważmy, że w *zachłanni* każdy z graczy ma wpływ tylko na swoją liczbę punktów. Ponadto od każdego wyboru gracza istnieje lepszy wybór (wybranie większej liczby), a jednocześnie nie istnieje żaden najlepszy wybór – nie istnieje przecież największa liczba. Spróbujcie zagrać z kimś w *zachłannych*. Zobaczycie, że na początku kłopotliwe będzie ustalenie zasad wyboru swojej liczby. Czy ograniczymy czas albo liczbę znaków do podania swojej liczby? Czy możemy wykorzystać zapis potęgowy do zapisania liczby czy musi być w postaci dziesiętnej? Czy musimy zapisać wybraną liczbę czy można ją powiedzieć?

Pomimo nasuwających się kolejnych pytań, zwiastujących poważne kłótnie pomiędzy graczami przy rozstrzyganiu zwycięzcy (tak naprawdę bywało), *zachłanni* z punktu widzenia teorii gier zasługuje na nazwanie *grą*.

Poznajmy jeszcze parę pojęć, które zawężą nam ilość gier w kręgu naszych zainteresowań.

Grą o sumie zero nazywamy grę, w której suma punktów, uzyskanych przez wszystkich graczy, jest zawsze równa zero. W przypadku dwuosobowych gier o sumie zero, obaj gracze mają przeciwstawne interesy: im więcej punktów zdobędzie jeden z graczy, tym mniej uzyska jego przeciwnik.

Grą z kompletną informacją nazywamy grę, w której każdy z graczy w momencie podejmowania decyzji ma pełną informację o decyzjach podjętych dotychczas przez pozostałych graczy, a także o wynikających z nich konsekwencjach dla każdego z graczy. Są to głównie gry, w których gracze wykonują ruchy na przemian, znając wybory, dokonane wcześniej przez przeciwników. Takie gry nazywamy *grami turowymi*.

Do dwuosobowych gier z kompletną informacją należą szachy, warcaby, go, gomoku (potocznie zwane jako kółko i krzyżyk). Grami z niekompletną informacją są między innymi papier - kamień - nożyce, poker, brydż oraz opisana wcześniej *zachłanni*.

W wielu popularnych grach jak szachy, warcaby czy go, reguły mówią, że za zwycięstwo uzyskuje się 1 punkt, za remis 0.5, a za przegraną 0. Wtedy suma punktów graczy jest równa 1. Takie gry są uogólnieniem gier o sumie zero i noszą miano gier o sumie stałej k (tutaj $k = 1$). Jeżeli w grze o sumie stałej każdemu z graczy po równo odejmiemy $\frac{1}{k}$ punktu, to uzyskamy grę o sumie zero. Stąd prawie wszystkie własności gier o sumie zero zachodzą dla gier o sumie stałej. Z wyżej wymienionych gier tylko *zachłanni* nie jest grą o sumie stałej.

W tym artykule skoncentrujemy się na chronologicznie uporządkowanych zastosowaniach najważniejszych algorytmów w szachach. Jednocześnie bądźmy świadomi faktu, że zaprezentowane metody bez większych zmian mogą być wykorzystane (i są) do tworzenia silnych programów, grających w dowolne inne, skończone dwuosobowe gry o sumie stałej i z kompletną informacją.

2. Proste algorytmy (Automat Quevedo)

W 1890 roku Leonardo Torres y Quevedo opisał proste algorytmy, prowadzące do wygranych w końcówkach szachowych typu biały król i wieża przeciw czarnemu królowi. Oryginalny algorytm prezentuje się następująco:

Definiujemy dwie strefy na szachownicy: pierwsza składa się z kolumn a, b i c, druga z kolumn f, g i h. Jeżeli czarny król:

1. *jest w tej samej strefie, co wieża, to wieża opuszcza tę strefę, wjeżdżając na kolumnę a lub h;*
2. *nie jest w tej samej strefie, co wieża, oraz pionowa odległość między czarnym królem a wieżą jest:*
   - *2.1 większa niż jedno pole, to wieża przesuwa się o jedno pole pionowo w kierunku czarnego króla;*
   - *2.2 jedno pole oraz pionowa odległość pomiędzy królami jest*
     - *2.2.1 większa niż dwa pola, to król przesuwa się o jedno pole pionowo w kierunku czarnego króla;*
     - *2.2.2 dwa pola oraz pozioma odległość pomiędzy królami jest*
       - *2.2.2.1 nieparzysta, to wieża (w swojej strefie) przesuwa się poziomo na kolumnę b lub g;*
       - *2.2.2.2 niezerowa parzysta, to król przesuwa się o jedno pole poziomo w kierunku czarnego króla;*
       - *2.2.2.3 zero, to wieża przesuwa się o jedno pole pionowo w kierunku czarnego króla.*

Działanie algorytmu możemy prześledzić na przykładzie, którego początkową pozycję przedstawia diagram 1.

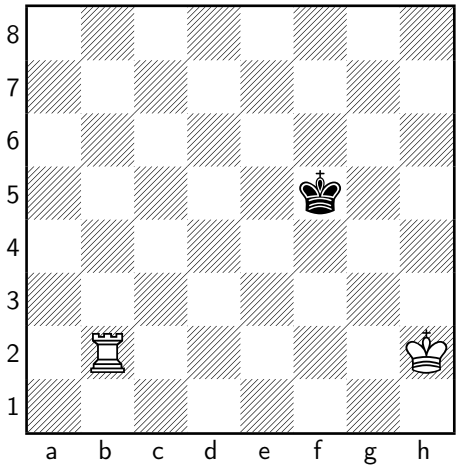


Diagram 1.

Musimy jednak nieco poprawić punkt *2.2.2.1* – w takiej formie występuje we wszystkich źródłach, jednak bez jego zmiany, 9. posunięcie białych w tym przykładzie nie byłoby określone przez algorytm. Zatem, niech:

2.2.2.1 nieparzysta, to wieża (w swojej strefie) przesuwa się poziomo na kolumnę b lub g, a jeżeli to niemożliwe, to na kolumnę a lub h;

Stosując algorytm Quevedo do najoptymalniejszych według silnika szachowego odpowiedzi czarnych, gra toczy się następująco: **1 ♖b3 ♔f4 2 ♔g2 ♔e4 3 ♔f2 ♔d4 4 ♔e2 ♔c4 5 ♖h3 ♔d4 6 ♖g3 ♔d5 7 ♖g4 ♔e5 8 ♔e3 ♔d5 9 ♖h4 ♔d6 10 ♖h5 ♔e6 11 ♔e4 ♔f6 12 ♖a5 ♔e6 13 ♖a6+ ♔e7 14 ♔e5 ♔d7 15 ♖b6 ♔c7 16 ♖h6 ♔d7 17 ♖g6 ♔c7 18 ♔d5 ♔d7 19 ♖g7+ ♔e8 20 ♔d6 ♔f8 21 ♖a7 ♔e8 22 ♖b7 ♔f8 23 ♔e6 ♔g8 24 ♔f6 ♔h8 25 ♔g6 ♔g8 26 ♖b8#**.

W 1912 roku Quevedo skonstruował maszynę, stosującą powyższy algorytm dla początkowych ustawień białych ♔a8 i ♖b7. Kolejna wersja, ulepszona przez jego syna, powstała w 1920 roku. Oryginalną maszynę, realizującą algorytm Quevedo i nadal działającą, można do dzisiaj podziwiać w Muzeum Torresa Quevedo w Madrycie.

Ze względu na ograniczone możliwości techniczne 100 lat temu, maszyna nie jest efektywna. W rzeczywistości w pozycji z diagramu 1 białe mogą zamatować czarne dużo szybciej, bo w 12 posunięciach. Algorytm Quevedo nie zadziała dla wielu pozycji: na diagramie 2, zgodny z algorytmem ruch **1 ♖h4** jest niemożliwy, a na diagramie 3 króle są w tym samym wierszu, na co algorytm nie jest przygotowany.

Uzupełnimy algorytm Quevedo tak, aby wskazywał drogę do mata dla dowolnego, początkowego ustawienia białej wieży i króla oraz czarnego króla. Zdefiniujemy najpierw zbiór

$$PATY = \{a3, b3, c3, c2, c1, a6, b6, c6, c7, c8, h3, g3, f3, f2, f1, h6, g6, f6, f7, f8\}.$$

Zauważmy, że jeżeli czarny król stoi na polu, nienależącym do $PATY$, to pozycja patowa jest niemożliwa.

Najpierw algorytm sprawdzi, czy jest patowa pozycja (ale przy ruchu białych). Wtedy czarny król stoi zawsze w rogu, biała wieża na o jedno pole na skos od niego, a biały król pilnuje swojej wieży. W takiej sytuacji algorytm doprowadzi do zamatowania czarnego króla w 3 posunięciach (*1.* punkt algorytmu).

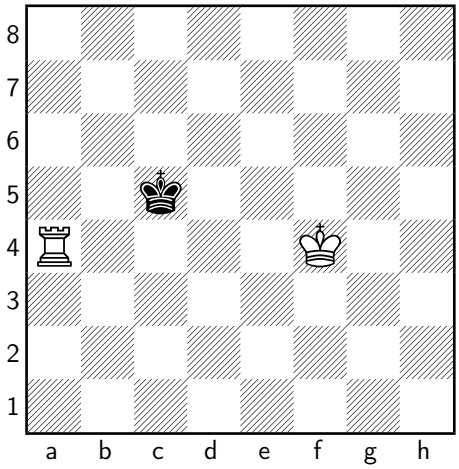


Diagram 2.

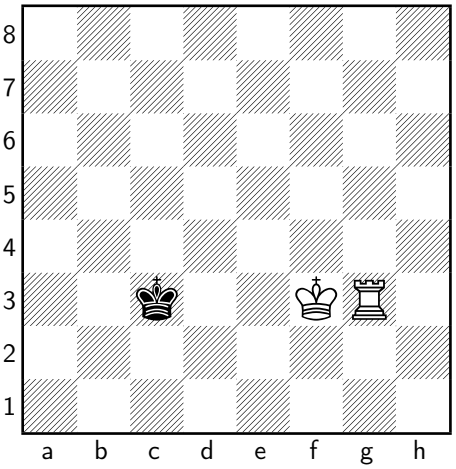


Diagram 3.

Potem, jeżeli biały król stoi na polu $PATY$, to każemy mu wyjść ze zbioru $PATY$, aby dalej uniknąć przypadkowego pata (*2.* punkt). Dalej zagwarantujemy sobie, że króle będą odseparowane linią, kontrolowaną przez wieżę (*3.* punkt). Po ustaleniu, że będzie to linia pozioma (choćby z nazwy), grę można kontynuować, używając algorytmu Quevedo.

1. *Jeżeli jest pozycja patowa, to wieża oddala się o jedno pole od bocznej linii, która jest najbliżej białego króla. Po ruchu czarnych, jeżeli biały król*

   1.1 *stoi o jedno pole od bocznej linii, to wieża przesuwa się na sąsiednie pole, będące odległe o 2 od każdej z bocznych linii, a w kolejnym ruchu wieża przesuwa się w przeciwnym kierunku co poprzedni o dwa pola (mat);*

   1.2 *nie stoi o jedno pole od bocznej linii, to przesuwa się (w pionie lub w poziomie) o jedno pole w kierunku czarnego króla, a w kolejnym ruchu wieża przesuwa się o jedno pole na boczną linię (mat);*

2. *Jeżeli pole, na którym stoi biały król, należy do $PATY$, to biały król przesuwa się na pole, nienależące do zbioru $PATY$.*

3. *Jeżeli wieża nie stoi na linii, rozdzielającej króle, to*

   3.1 *jeżeli wieża nie może stanąć na skos obok białego króla, to przesuwa się na pole, nieatakowane przez czarnego króla i należące do linii, sąsiadującej z królem; następnie przesuwa się na pole na skos obok białego króla.*

   3.2 *Dopóki wieża nie stoi na linii, rozdzielającej króle, wieża przesuwa się na kolejne pole na skos obok białego króla, na którym jeszcze nie stała w tym algorytmie.*

4. *Jeżeli wieża nie stoi w wierszu, rozdzielającym króle, to formalnie zmieniamy nazewnictwo i oznaczenia: wierszy na kolumny i kolumn na wiersze.*

5. *Jeżeli wieża nie stoi ani na kolumnie a, ani h, to przesuwa się na jedną z tych kolumn, na pole, które nie jest atakowane przez czarnego króla. Dalej stosujemy algorytm Quevedo.*

Zastosujmy nasz algorytm do pozycji z diagramu 3, ponownie grając przeciw popularnemu silnikowi szachowemu. Przy każdym posunięciu białych jest odnośnik do wykorzystywanej w tym posunięciu instrukcji algorytmu.

1 ♔f4+ ***(2.)*** **♔d4 2 ♖e3** ***(3.2.)*** **♔d5 3 ♖e1** ***(4.) oraz (5.)*** Dalej stosujemy algorytm Quevedo: **3. . . ♔d6 4 ♔f5** ***(2.2.2.2.)*** **♔d5 5 ♖d1+** ***(2.2.2.3)*** **♔c4 6 ♔e5** ***(2.2.1)*** **♔c3 7 ♖d8** ***(1.)*** **♔c4 8 ♖d7** ***(2.2.2.1)*** **♔b5 9 ♖c7** ***(2.1.)*** **♔b6 10 ♖c1** ***(1.)*** **♔b5 11 ♔d5** ***(2.2.1)*** **♔b4 12 ♖c2** ***(2.2.2.1)*** **♔b3 13 ♖c8** ***(1.)*** **♔b4 14 ♖c7** ***(2.2.2.1)*** **♔b3 15 ♔d4** ***(2.2.2.2.)*** **♔b4 16 ♖b7+** ***(2.2.2.3)*** **♔a5 17 ♔c4** ***(2.2.1.)*** **♔a6 18 ♖b1** ***(2.1.)*** **♔a5 19 ♖b2** ***(2.2.2.1.)*** **♔a6 20 ♔c5** ***(2.2.2.2.)*** **♔a7 21 ♔c6** ***(2.2.2.2.)*** **♔a8 22 ♔c7** ***(2.2.2.2.)*** **♔a7 23 ♖a2#** ***(2.2.2.3.)***

Nasz algorytm nadal jest daleki od doskonałości. W paru miejscach nie wyznacza jednoznacznie pola, na które powinny pójść białe figury, bazując tylko na tym, że istnieje ruch białych, spełniający zadane warunki. Nic dziwnego, że maszyna Quevedo, całkiem skomplikowana, nie doczekała się wzbogacenia o jeszcze lepszy algorytm. Musiało upłynąć kilkadziesiąt lat, aby nauka była na to gotowa. Podejście do wyszukiwania najlepszych posunięć też się zmieniło.

Zadanie 1. Spróbuj wymyślić i zapisać w podobny sposób algorytm, prowadzący do zamatowania czarnego króla dla dowolnej pozycji na szachownicy z białym hetmanem i królem oraz czarnym królem. Następnie poproś kogoś, aby przetestował Twój algorytm pod kątem działania dla każdej wyjściowej pozycji. Bardzo prawdopodobne, że wnikliwy tester znajdzie lukę w algorytmie. Jeżeli mu się to nie uda, to brawo Ty!

3. Algorytm min-maks i alfa-beta

Nie zdziwię się, jeżeli czytelnik stwierdził, że algorytm z poprzedniego rozdziału jest dość rozwlekły jak na „tylko" matowanie królem i wieżą. Średnio zaawansowany amator szachów potrafi zamatować wieżą w dość intuicyjny sposób. Jednak wytłumaczenie takiego sposobu człowiekowi zawiera ogólniki, zrozumiałe dla ludzi, ale nie mogące posłużyć do opisu algorytmu. Nieco zmieniony powyższy algorytm w postaci bardziej „ludzkiej" (jednak nieodpowiedniej do zaimplementowania ze względu na brak precyzji w formułowaniu poleceń) może brzmieć:

1. *Nigdy nie wykonaj ruchu, po którym jest pat.*
2. *Wjedź wieżą na linię między królami.*
3. *Jeśli czarny król zaatakuje Twoją wieżę, to odjedź wieżą jak najdalej od niego, nie opuszczając linii, oddzielającej króle.*
4. *Wymuś, aby jego król stanął naprzeciw Twojego – wtedy szachem wieżą odcinasz czarnemu królowi kolejną linię.*
5. *Jeśli nie możesz tego wymusić (bo czarny król „ucieka" od białego), to wykonaj ruch wieżą na linii, oddzielającej króle, ustawiając wieżę jak najdalej od czarnego króla.*

Brzmi bardziej treściwie niż algorytm z poprzedniego rozdziału lecz jest to zbiór porad, a nie algorytm. Natomiast algorytm Quevedo jest adaptacją takiego opisu, będącego naśladowaniem ludzkiego postępowania, na język algorytmów.

Kolejne podejścia do algorytmów szachowych i ich realizacji wykorzystywały możliwości coraz szybszych komputerów, a jednocześnie powoli zaczynały odchodzić od schematów ludzkiego rozumowania. Gdy komputery były już dostatecznie szybkie, to do rozgrywania prostych końcówek wystarczyła metoda „brute force", polegająca na odpowiednim przeszukaniu wszystkich możliwych kontynuacji gry aż do konkluzji, że cokolwiek nie uczyni przeciwnik, i tak program da radę go zamatować. Również pojawienie się mocnych narzędzi matematycznych ułatwiło odpowiedź na pytanie, jak efektywniej szukać najlepszego posunięcia, przeszukując możliwe kontynuacje gry.

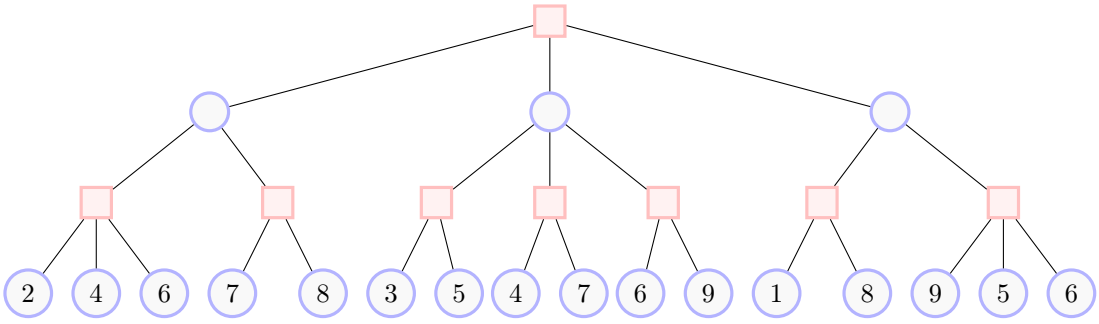
W 1928 roku John von Neumann udowodnił twierdzenie o istnieniu „najlepszych" strategii w bardzo obszernej rodzinie gier [4]. Przedstawimy wersję tego twierdzenia, ograniczającą się do gier o kompletnej informacji i skończonej liczbie wyborów. [1]

Twierdzenie 1. *(von Neumanna o minimaksie): dla dowolnej dwuosobowej gry z kompletną informacją, skończoną liczbą wyborów i o sumie równej zero istnieje liczba rzeczywista v (zwana wartością gry), dla której:*

- gracz pierwszy może zagrać w taki sposób, że liczba zdobytych przez niego punktów będzie nie mniejsza niż v,

- gracz drugi może zagrać w taki sposób, że liczba zdobytych przez niego punktów będzie nie mniejsza niż $-v$.

Wyznaczanie najlepszej kontynuacji dla każdego z graczy w grach turowych prześledzimy na przykładzie gry, którą opisuje rysunek 1. Dla wygody, nazwijmy ją *takasobiegra*.



Rysunek 1. Pierwszy gracz zaczyna grę w górnym kwadracie i wybiera któreś z poniższych kółek. Potem gracz drugi wybiera z tego kółka kontynuację do któregoś z niższych kwadratów, wreszcie gracz pierwszy wybiera z poprzednio wybranego kwadratu drogę do któregoś z poniższych kółek. Wtedy gracz pierwszy zdobywa liczbę punktów, równą liczbie w wybranym kółku, a gracz drugi tyle samo punktów traci.

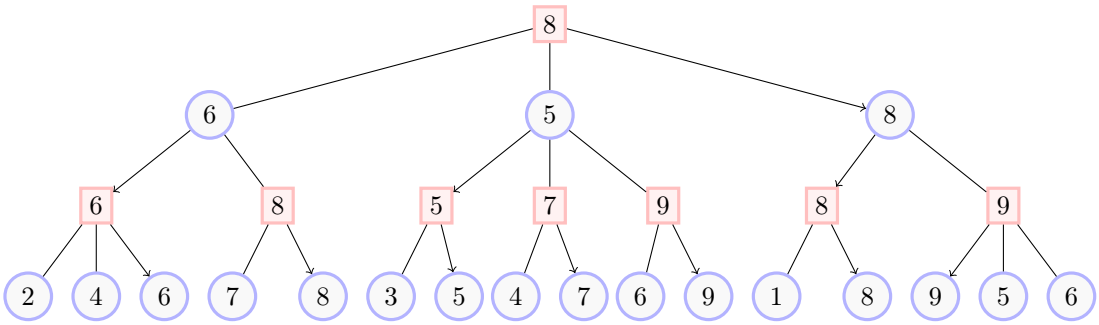
Jest to turowa, dwuosobowa gra o sumie zero. Z twierdzenia o minimaksie wynika, że dla *takiejsobiegry* istnieje liczba rzeczywista v równa liczbie punktów, którą może sobie zagwarantować gracz pierwszy, a jednocześnie gracz drugi może sobie zagwarantować liczbę punktów równą $-v$. Znajdziemy ją oraz najkorzystniejsze wybory graczy, stosując algorytm o nazwie min-maks. Używa się też nazwy analiza wsteczna, gdyż rozpoczniemy analizę „od końca" gry.

Twierdzenie 2. *(Algorytm min-maks) Mamy dane drzewo, definiujące dwuosobową grę turową ze skończoną liczbą wyborów i o sumie zero, a także z ustaloną liczbą punktów, jaką zdobywa gracz pierwszy*

[1] W ten sposób unikamy wprowadzania niepotrzebnych tutaj pojęć, takich jak strategia mieszana czy wartość oczekiwana gry.

w każdym wierzchołku, kończącym grę. W każdym wierzchołku, w których decyduje gracz 1, piszemy największą liczbę z wierzchołków, występujących bezpośrednio pod tym wierzchołkiem. Jeżeli wierzchołek należy do gracza drugiego – najmniejszą. Wtedy liczba v, zapisana w najwyższym wierzchołku, jest równa wartości gry. Gracze pierwszy i drugi, wybierając w czasie gry kontynuacje, zgodne z wyborami w algorytmie min-maks, zagwarantują sobie co najmniej odpowiednio v i $-v$ punktów.

Efekt zastosowania algorytmu min-maks do *takiejsobiegry* przedstawia rysunek 2.



Rysunek 2. Liczba 8 z najwyższego wierzchołka to wartość gry. Gracz pierwszy i drugi, decydując w czasie gry na poruszanie się wzdłuż strzałek, zapewniają sobie co najmniej odpowiednio 8 i -8 punktów.

Teoretycznie możemy algorytm min-maks zastosować do rozwiązania wszelkich gier turowych o sumie zero (rozwiązanie gry to wyznaczenie najlepszych posunięć oraz wyniku gry, w której gracze stosowali najlepsze posunięcia). Jednak w przypadku bardziej złożonych gier (a do takich szachy należą) pojawia się problem: nie potrafimy przy obecnym stanie techniki wygenerować wszystkich możliwych kontynuacji na wzór rysunku 2. Już po jednym ruchu białych i czarnych jest 400 możliwych pozycji na szachownicy, z kolejnymi ruchami, liczba pozycji do przeanalizowania gwałtownie rośnie.

Do rozwiązania gry szachy jest jeszcze daleka droga. Na chwilę obecną (2023 rok) istnieje baza wszystkich pozycji szachowych, zawierających co najwyżej 7 figur lub pionów (jest ich 423836835667331) z informacją, kiedy i który z graczy może zapewnić sobie wygraną w każdej z tych pozycji. Nie zanosi się na szybkie „rozgryzienie" wszystkich pozycji z 8 bierkami na szachownicy, nie mówiąc już o 32 bierkach, od których zaczyna się każda partia szachów.

Claude Shannon oszacował liczbę wszystkich możliwych partii na 10^{120} [5]: załóżmy, że każda partia trwa średnio 40 posunięć oraz po każdym posunięciu jest średnio 10^3 kontynuacji po jednym ruchu białych i czarnych. Stąd $(10^3)^{40} = 10^{120}$. Jest to bardzo grube oszacowanie, jednak daje nam wyobrażenie, jak szerokie musiałoby być drzewo obrazujące wszystkie możliwe partie szachowe.

Samo drzewo gry w szachy jest skończone dzięki regule 50 posunięć. Jeżeli w 50 kolejnych posunięciach białych i czarnych nie nastąpi bicie ani ruch pionem, to partia zakończy się remisem. Liczba możliwych bić i ruchów pionami jest skończona, a najdłuższa możliwa partia szachowa liczy 5950 posunięć obu graczy.

Wprawdzie ilość wszystkich możliwych partii szachowych znacznie przerasta obecne możliwości obliczeniowe komputerów, jednak algorytm min-maks można zastosować w szachach i to w każdej pozycji. Należy go nieco zmodyfikować, aby działał na drzewach takiej wielkości, która pozwoli wybrać najlepszą kontynuację w miarę przyzwoitym czasie. Jednym ze sposobów jest „obcięcie" drzewa, reprezentującego grę, do odpowiedniej wysokości. Problemem jest wtedy przypisanie wartości w miejscach ucięcia drzewa. Wszak odpowiadają im pozycje w grze, w których wynik nie jest jeszcze znany. Musimy wtedy jak najlepiej zdefiniować *ocenę pozycji*. Jest to jedno z największych wyzwań przy tworzeniu większości współczesnych

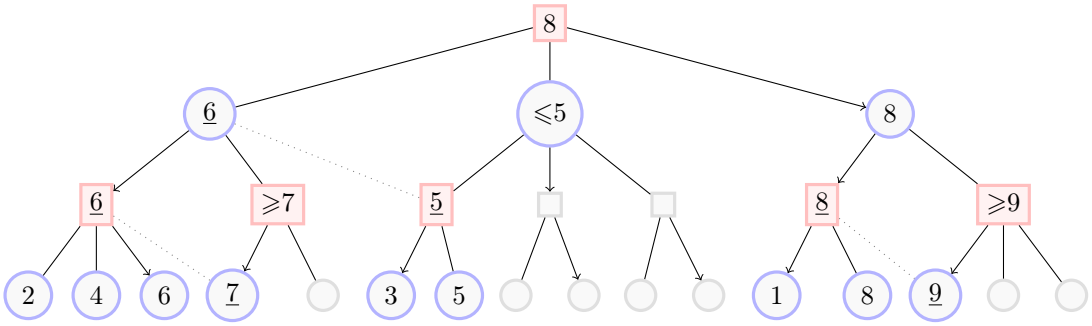
algorytmów szachowych. Ocena pozycji na podstawie samej przewagi materialnej jednej czy drugiej strony może być nieadekwatna do faktycznej sytuacji na szachownicy. Ponadto, jeżeli drzewo, reprezentujące grę, zostało ucięte w trakcie wymiany silnych figur, ocena pozycji na podstawie przewagi materialnej może być błędna, jeżeli w kolejnym, obciętym już ruchu, następuje zakończenie wymiany figur. W przypadku czołowych silników (programów) szachowych, algorytmy na różne sposoby radzą sobie z oceną pozycji, o czym będzie jeszcze mowa.

Przypuśćmy, że już udało nam się ustalić kryterium oceny pozycji. Inaczej: załóżmy, że jesteśmy w stanie wybranym pozycjom szachowym przyporządkować liczbę rzeczywistą – im większą, tym większa jest przewaga białych. Standardowo, silniki szachowe przyjmują 0 dla pozycji remisowej, 1 odpowiada przewadze piona, a 1000, jeżeli doliczą się wygranej białych (-1000 – czarnych). Wtedy algorytm min-maks można zastosować. Znalezione optymalne według niego ruchy wcale nie muszą być najlepszymi z powodu możliwych niedoskonałości oceny pozycji. Praktycznie każdy silnik szachowy ma na swoim koncie przegraną.

Z czasem algorytm min-maks został ulepszony do algorytmu alfa-beta w taki sposób, aby wykluczyć przegląd niektórych zbędnych gałęzi w drzewie, opisującym grę. Z tego powodu algorytm alfa-beta w tym samym czasie może znacznie głębiej przeanalizować kontynuacje niż w min-maks. Przedstawiając ten algorytm, będziemy używać nomenklatury rodem z drzew genealogicznych: pojawią się wujek, syn czy ojciec.

Twierdzenie 3. *(Algorytm alfa-beta) Mamy dane drzewo, definiujące dwuosobową grę turową ze skończoną liczbą wyborów i o sumie zero, a także z ustaloną liczbą punktów, jaką zdobywa gracz pierwszy w każdym wierzchołku, kończącym grę. Jeżeli w trakcie stosowania algorytmu min-maks, w pewnym wierzchołku będzie zapisana mniej korzystna wartość dla gracza, decydującego w tym wierzchołku, niż w pewnym bratanku tego wierzchołka, to możemy ominąć w algorytmie analizę części drzewa, zawierającej wszystkich braci tego bratanka wraz z ich potomkami.*

Powróćmy do *takejsobiegry* i przeprowadźmy algorytm alfa-beta z lewej do prawej strony drzewa.



Rysunek 3. Algorytm alfa-beta, przeprowadzony od lewej do prawej strony drzewa.

Na rysunku 3 szarym kolorem zaznaczone zostały wierzchołki, które w algorytmie alfa-beta mogą być pominięte.

Wierzchołki, dla których przypisana wartość liczbowa w trakcie stosowania algorytmu min-maks jest mniej korzystna dla gracza decydującego w tym wierzchołku niż w jego bratanku, połączone zostały przerywaną linią. Przykładowo, podkreślona szóstka w kwadracie dla decydującego w nim gracza pierwszego jest mniej korzystna, niż podkreślona siódemka w kółku (bratanku owego kwadrata z szóstką). Stąd

na pewno ojciec kółka z siódemką (czyli kwadrat z oznaczeniem $\geqslant 7$) będzie mieć wartość co najmniej siedem, a jego ojciec (koło z podkreśloną szóstką) dokładnie sześć. Ta wartość nie zależy od wartości, przyporządkowanej drugiemu synowi kwadrata z $\geqslant 7$, więc nie trzeba jej odczytywać.

Alfa-beta jest algorytmem, który w grach z dużą liczbą możliwych wyborów znacznie szybciej odnajduje kontynuację, prowadzącą do korzystnego wyniku. Dlatego jego zaawansowane wersje były sercem wielu czołowych programów szachowych.

Zadanie 2. Przeprowadź algorytm alfa-beta w *takiejsobiegrze* z prawej do lewej strony drzewa.

4. Dlaczego Stockfish jest tak silny?

Właściwie w tytule rozdziału mógłby się pojawić dowolny inny, popularny program szachowy. Opisywane dalej fakty mogą się odnosić także do nich. Skoncentrujemy się jednak na jednym z najczęściej wykorzystywanych obecnie silników szachowych, jakim jest Stockfish. Różne jego wersje nie mają dużych wymagań sprzętowych. Jest to wolne oprogramowanie, na bieżąco doskonalone przez licznych programistów przy pomocy platformy Fishtest. Obecnie ponad 100 osób miało znaczący wpływ na ulepszenie kodu.

Z poprzedniego rozdziału wiemy, że w tworzeniu optymalnego algorytmu szachowego podstawowym problemem jest ustalenie właściwej wartości wierzchołków w ograniczonym drzewie, symbolizującym grę.

Poniżej przedstawione zostały wybrane założenia, jakimi kierowali się początkowo autorzy Stockfisha przy ocenie pozycji [6]:

- przewaga materialna,
- kontrola centralnych pól szachownicy,
- obrona własnych figur,
- atakowanie figur przeciwnika,
- unikanie posiadania izolowanych pionów [2],
- posiadanie gońca lub gońców na przekątnych szachownicy,
- zasłonięcie króla własnymi bierkami.

Tych założeń jest znacznie więcej i są bardziej szczegółowe. Ale jak ustalić, które są ważniejsze, a które mniej? Kluczem do sukcesu jest optymalne ustalenie funkcji, która na podstawie wszystkich założeń zwróci najodpowiedniejszą ocenę pozycji.

Z pomocą przychodzą również algorytmy heurystyczne, czyli poszukujące korzystnych rozwiązań bez gwarancji znalezienia najlepszego. W szachach określenie, która z kontynuacji jest najlepsza, nie zawsze jest jednoznaczne, więc bazuje także na prawdopodobieństwie dojścia do sukcesu. Stockfish używa również funkcji, optymalizujących wybory. Między innymi, kontynuacje, które są ocenione jako najbardziej obiecujące (choćby na podstawie wymienionych poprzednio założeń), są jeszcze głębiej przeglądane pod kątem poprawności. Jeżeli wykonałeś zadanie 2 i zastosowałeś algorytm alfa-beta dla gry z rysunku 1, wędrując po drzewie z prawej do lewej strony, to zauważyłeś, że algorytm odrzuci z analizy część drzewa z dolnymi, sąsiadującymi wierzchołkami z wartościami 3, 5 i ich ojcem. Zatem spory wpływ na efektywność algorytmu alfa-beta ma takie ustalenie kolejności przeszukiwania drzewa gry, aby zwiększyć prawdopodobieństwo odrzucenia analizy jak największej liczby wierzchołków drzewa.

[2] jeżeli pion nie posiada na sąsiednich kolumnach pionów tego samego koloru, to nazywa się izolowanym

Oprócz bardzo zaawansowanych algorytmów, Stockfish posiada dodatkowo biblioteki: debiutów, czyli bazę początkowych posunięć, uznanych za korzystne, a także wybranych końcówek. W najlepszych wersjach programu pojawiają się bazy wszystkich końcówek, posiadających co najwyżej 7 bierek na szachownicy, z określonym optymalnym sposobem postępowania.

Okazuje się, że nawet silne programy mają swoje słabe strony. Istnieją niezbyt skomplikowane pozycje na szachownicy, których ocena przez czołowe programy jest błędna.

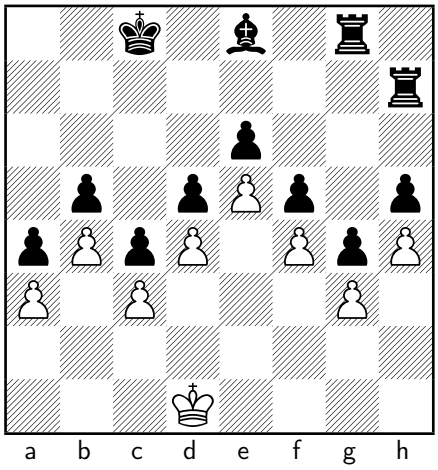


Diagram 4. Ruch białych.

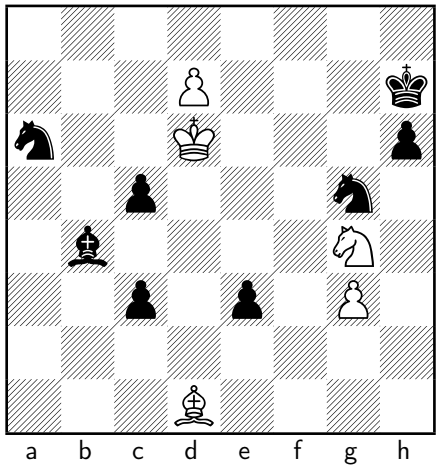


Diagram 5. Ruch białych.

W pozycji z diagramu 4 plansza jest wyraźnie podzielona łańcuchem pionowym na dwie części. W jednej porusza się biały król, a w drugiej czarne figury. Białe bez problemu mogą sobie zagwarantować remis. Wystarczy, że będą wykonywać dowolne ruchy królem, gdyż czarne nie są w stanie przedostać się na dolną część szachownicy. Natomiast wszystkie popularne silniki szachowe nie potrafią tej pozycji prawidłowo ocenić. Nie są zdolne do „ludzkiej" dedukcji, którą przeprowadziliśmy. Dla nich na diagramie jest przytłaczająca przewaga czarnych. Ze względu na sporą liczbę możliwych posunięć ze strony czarnych, drzewo obrazujące grę z tej pozycji jest bardzo szerokie, przez co głębokość poszukiwań programu wraz z czasem rośnie bardzo wolno.

Przedstawiona na diagramie 4. pozycja nosi miano zamkniętej – bezpośrednia interakcja pomiędzy bierkami obu stron jest znikoma. Próbują to wykorzystać szachiści, którzy chcą uzyskać korzystny wynik w pojedynku z szachowymi silnikami i wybierają warianty zamknięte. W pozycjach otwartych łatwiej o przeoczenie jakiegoś posunięcia przez człowieka, a takich błędów bezduszny program nie wybacza.

Diagram 5. przedstawia kompozycję szachową zwaną zadaniem Plasketta. Została ona utworzona przez Gijs van Breukelena około 1970 roku. Stała się słynna dzięki angielskiemu arcymistrzowi, J. Plaskettowi, który zademonstrował to zadanie w kuluarach silnego turnieju szachowego w Brukseli w 1987 roku. W pozycji z diagramu białe mają jedną, zwycięską kontynuację. Zadanie rozwiązał wtedy prawidłowo tylko jeden szachista, były Mistrz Świata, Michaił Tal.

Jest to zadanie, które sprawia ogromną trudność silnikom szachowym ze względu na konieczne do rozwiązania „ludzkie" rozumowanie. Bez niego mnogość wariantów znacznie utrudnia znalezienie rozwiązania nawet najsilniejszym programom. Szczegółowa analiza tej pozycji przez dwa różne programy została opisana w [3]. Pierwszym był Stockfish 14. Na procesorze 2.8 GHz Intel Core i5, po 1 minucie i 17 sekundach i przeszukaniu drzewa gry do głębokości 30 półruchów (czyli 15 ruchów białych i 15 czarnych),

wśród pięciu najwyżej ocenionych pierwszych posunięć nie było zwycięskiego posunięcia (jest nim **1 ♘f6**), a pozycja była oceniona na dającą duże szanse czarnym na wygraną! Dopiero po zwiększeniu głębokości poszukiwań do 39 półruchów, po 18 minutach i 6 sekundach program znalazł ruch **1 ♘f6** i ocenił go jako przynoszący zwycięstwo białym (no nie do końca; jeżeli mamy być bardzo dokładni, to był to ruch, który według programu był najlepszym dla białych z prawdopodobieństwem wygranej równym 90.9%).

Drugim programem, testującym tę pozycję, był będący wolnym oprogramowaniem Leela Chess Zero. Był on odpowiedzią na pojawienie się AlphaZero, działającym na podobnej zasadzie: wykorzystania sieci neuronowych i samodzielnego uczenia się poprzez rozgrywanie niezliczonej liczby partii, połączonym z ustalaniem wartości poszczególnych pozycji. Po przeanalizowaniu około 60 milionów pozycji, program ten, podobnie jak początkowo Stockfish 14, nie był w stanie znaleźć zwycięskiej kontynuacji. Co ciekawe, po wskazaniu pierwszego posunięcia, Leela Chess Zero uznał ruch **1 ♘f6** za najlepszy (i prawidłowo ocenił go jako przynoszący zwycięstwo) po analizie 5.5 miliona pozycji. Aby dojść w pozycji z diagramu do tego samego wniosku, Stockfish 14 musiał przeanalizować aż 500 milionów pozycji. Stąd widać, że algorytm wykorzystujący sieci neuronowe Leela Chess Zero bardziej efektywnie przeszukuje drzewa gry, niż Stockfish 14. Schemat działania takich programów szachowych zostanie przybliżony w kolejnym rozdziale.

A my zobaczmy, jak wygląda rozwiązanie zadania z diagramu 5:

1 ♘f6+ ♔g7

Czarne wykonały jedyny ruch, który utrudnia dorobienie białym hetmana na polu d8, grożąc ruchem 2... ♘f7. Po 1... ♔g6 2 ♗h5+ i następnie można spokojnie dorobić hetmana.

2 ♘h5+ ♔g6 Jeżeli 2... ♔h7 to 3 ♗c2+ spycha czarnego króla na ostatnią linię, co ponownie umożliwia dorobienie hetmana bez obawy o jego stratę.

3 ♗c2+!! Ten ruch jako kontynuację gry na wygraną przegapia większość silników szachowych.

3... ♔×h5 4 d8♕!! a jednak! **4... ♘f7+**

Naturalna odpowiedź dla człowieka – zlikwidujmy hetmana, inaczej białe będą mieć materialną przewagę. Tutaj musimy zwrócić honor silnikom szachowym, które odkryły, że czarne mają silniejsze kontynuacje, niż 4... ♘f7. Między innymi, takie posunięcia, jak 4... ♔g4, 4... c4+ czy 4... e2 nie dają widoków na szybkie zamatowanie czarnych, a jedynie na znaczną przewagę materialną.

5 ♔e6 ♘×d8+ 6 ♔f5

Dalszy plan białych polega na stwarzaniu gróźb zamatowania czarnego króla swoim gońcem. Optymalna gra czarnych to jedynie maksymalne opóźnianie tego mata. W głównym wariancie są aż dwie słabe promocje ze strony czarnych – to kolejna przyczyna złej oceny tego zadania przez silniki szachowe. Ponieważ te promocje nie kończą się zamatowaniem przez czarne, więc algorytmy przeszukujące chętniej analizowały poddrzewa, w których istniała promocja na hetmana, dająca znacznie więcej możliwych posunięć czarnym.

6... e2 7 ♗e4 e1♘ 8 ♗d5 c2 9 ♗c4 c1♘ 10 ♗b5 ♘c6 11 ♗×c6 ♘c7 12 ♗a4

Biały goniec podąża już na pole d1, z którego dojdzie do ostatecznego ataku matującego.

Zadanie 3. Spróbuj sam ułożyć pozycje na szachownicy, które będziesz w stanie ocenić pod kątem przewagi jednej ze stron, a z którymi nie poradzi sobie któryś z dostępnych programów szachowych.

5. Dlaczego AlphaZero jest jeszcze silniejszy?

AlphaZero jest programem, utworzonym przez przedsiębiorstwo DeepMind i pracującym na pojedynczej sieci neuronowej typu Resnet. Wykorzystuje on sztuczną inteligencję do samodzielnego uczenia się grania w różne gry turowe, w tym w szachy.

W przypadku szachów, w zależności od pozycji na szachownicy, algorytm wyznacza ocenę (wartość) tej pozycji oraz prawdopodobieństwo wyboru każdego z posunięć. Ocena pozycji zależy od wyników losowo granych partii, które się w niej rozpoczęły. Udoskonalenie wyznaczonych przez algorytm wyników następuje przy wykorzystaniu w odpowiedni sposób powszechnej w modelowaniu matematycznym metody Monte Carlo.

1. W ustalonym drzewie gry, najpierw wybieramy „najbardziej obiecujący wierzchołek". Na początku może być to dowolnie wybrany wierzchołek, jednak po przeprowadzeniu większej liczby symulacji, wybór „najbardziej obiecującego wierzchołka" do kolejnej symulacji podlega ustalonym regułom, o których za chwilę będzie mowa.

2. Następuje rozegranie losowej partii (w bardzo zaawansowanych algorytmach nie do końca losowej) z tej pozycji.

3. Wynik rozegranej partii zmienia ocenę wierzchołków, przez które przechodziła gra.

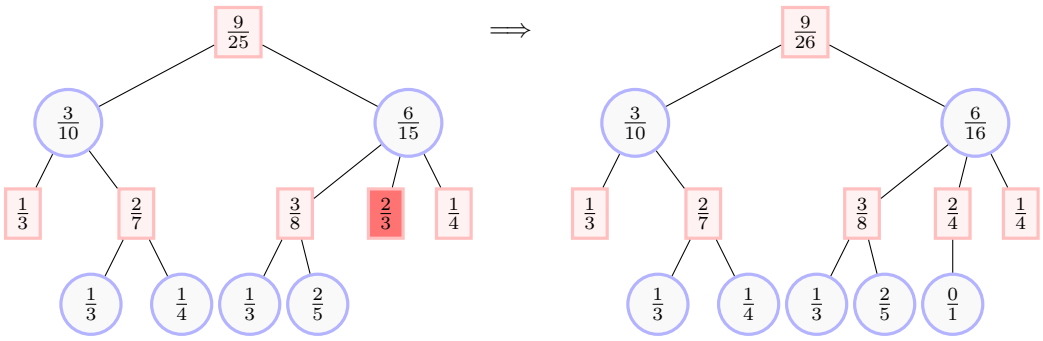
Im więcej symulacji zostanie przeprowadzonych, tym większa szansa na uzyskanie najlepszego rozwiązania.

Prześledźmy wykonanie jednej symulacji gry w czasie stosowania metody Monte Carlo. Przypuśćmy, że dotychczasowy efekt jej stosowania przedstawia lewe drzewo na rysunku 4. Ułamek w każdym wierzchołku jest ilorazem liczby symulacji gry z tego wierzchołka zakończonych zwycięstwem pierwszego gracza i liczby wszystkich symulacji, poprowadzonych przez ten wierzchołek.

Załóżmy, że algorytm wybrał „najbardziej obiecującym wierzchołkiem" wierzchołek, zaznaczony na czerwono.

Następnie wybierany jest losowo syn obiecującego wierzchołka, po czym program przeprowadza losową symulację z tego wierzchołka.

Załóżmy, że symulacja zakończyła się zwycięstwem gracza drugiego. Wtedy każdy wybrany uprzednio syn oraz wszyscy jego przodkowie uzyskują aktualizację swoich wartości, zgodnie z wynikiem zakończonej symulacji (prawe drzewo na rysunku 4).



Rysunek 4. Pojedyncza symulacja w metodzie Monte Carlo.

Wybór „najbardziej obiecującego wierzchołka", od którego przeprowadzona będzie kolejna symulacja, jest kompromisem między wyborem wierzchołków mocno *wyeksploatowanych* w przeprowadzonych już symulacjach i z wyższą oceną, a wierzchołków słabo *wyeksplorowanych*. Wyznaczona wartość wierzchołka może być daleka od właściwej z powodu zbyt małej liczby symulacji, przechodzących przez ten wierzchołek. Dlatego wybór kolejnego wierzchołka do symulacji określa specjalna funkcja, przyporządkowująca każdemu wierzchołkowi w pewną wartość. Wierzchołek o najwyższej wartości jest wybierany do kolejnego kroku algorytmu Monte Carlo. Jedną ze stosowanych w praktyce funkcji jest wprowadzona w [1]:

$$f(w) = \frac{p(w)}{s(w)} + c \cdot \sqrt{\frac{\ln n(w)}{s(w)}}.$$

Tutaj $s(w)$ oznacza liczbę symulacji partii, przeprowadzonych przez wierzchołek w, $p(w)$ to suma punktów, uzyskanych w tych symulacjach przez gracza, decydującego w wierzchołku w. $n(w)$ to liczba przeprowadzonych symulacji, przechodzących przez ojca wierzchołka w, a c jest stałą, określaną w zależności od istniejących warunków. To czy wartość funkcji $f(w)$ będzie duża, zależy zarówno od odsetka wygranych dalej partii (część $\frac{p(w)}{s(w)}$), jak i od tego, czy stosunkowo często był pomijany wierzchołek w w symulacjach (część $c \cdot \sqrt{\frac{\ln n(w)}{s(w)}}$).

Zatem udział losu w poszukiwaniu najlepszych kontynuacji, który się także pojawiał w programach typu Stockfish, w AlphaZero odgrywa główną rolę. To paradoksalne, bo przecież gra szachy nie posiada żadnej losowości w swoich zasadach.

Przed pierwszym uruchomieniem, AlphaZero znał tylko same zasady gry w szachy. Żadnych bibliotek debiutowych, końcówek, wprowadzonych wartości pozycji... zaczynał się uczyć od zera! Następnie AlphaZero zaczął grać ze sobą partie szachowe. Pierwsza była całkowicie losowa, lecz wybór posunięć w kolejnych następował już przy użyciu metody Monte Carlo i tworzeniu ocen coraz większej liczby pozycji na szachownicy. Oceny te wraz ze wzrostem liczby zagranych partii były korygowane. Ten proces mógł się toczyć miesiącami. Im dłużej – tym silniejszy się stawał sam program.

Na pewno znane są czytelnikowi sposoby oznaczania kolejnych wersji programów. Najczęściej są to liczby – słyszeliśmy o Windows 11, Stockfish 15, Iphone 15 czy Python 33.11.4. W przypadku AlphaZero parametrem jest... czas samodzielnego uczenia się!

Przykładowo, AlphaZero po dziewięciogodzinnym uczeniu się zagrał stupartiowy mecz z programem Stockfish 8 w 2017 roku. Wygrał 28 partii, 72 zremisował i nie odniósł żadnej porażki. Było to przełomowe zwycięstwo, znak nowej epoki w tworzeniu programów szachowych. W czasie tego pojedynku, w ciągu jednej sekundy AlphaZero oceniał około 80 000 pozycji, gdy „tradycyjne" silniki szachowe przeszukują około 1000 razy więcej. AlphaZero czyni to jednak skuteczniej, wykorzystując strukturę sieci neuronowych w celu lepszej selekcji korzystnych kontynuacji. AlphaZero nie korzystał z osiągnięć ludzkości. On sam utworzył w ekspresowym tempie swoją historię szachów oraz wniósł wiele nowinek do dotychczasowej oceny wielu wariantów. Jego gra, nieintuicyjne poświęcenia czy na pozór niezgodne ze zdrowym rozsądkiem manewry figur nie przypominały nie tylko gry ludzi, ale również dotychczasowych programów szachowych.

Między innymi, AlphaZero stosunkowo rzadko wykonywał roszadę. Nie znaczy to jednak, że roszada jest niekorzystnym posunięciem. Nadal każdy szachista powinien pomyśleć o roszadzie celem zabezpieczenia króla i wprowadzenia wieży do gry (ludzkie podejście, prawda?) lecz AlphaZero potrafi znacznie lepiej przewidzieć i zapobiec ewentualnym problemom, wynikającym z braku roszady.

AlphaZero okazał się więc niezwykle efektywnym programem. Jego architektura jest przygotowana do wprowadzenia zasad dowolnej, kombinatorycznej gry, więc kwestią czasu było osiągnięcie spektakularnych

wyników w innych popularnych grach. Gra Go uchodzi za znacznie bardziej złożoną kombinatorycznie od szachów i na zwycięstwa programów komputerowych z czołowymi graczami świat musiał czekać do 2016 roku, gdy odmiana AlphaGoZero wygrała z koreańskim mistrzem Lee Sedol 4-1.

6. Podsumowanie

Nie wgłębialiśmy się zbytnio w szczegóły działania algorytmów szachowych. Ważniejsze było dla nas zobrazowanie metod ich tworzenia oraz działania. A te zmieniały się razem ze wzrostem technicznych możliwości na przestrzeni lat.

Najpierw był pionierski w dziedzinie implementacji (o ile można tak powiedzieć o budowie analogowej maszyny, bardziej rodem ze steampunkowych klimatów) algorytm Quevedo.

Późniejsze programy, wykorzystując moc komputerów, skupiały się na przeszukaniu jak największych drzew, obrazujących wszystkie możliwe posunięcia do ustalonego wcześniej poziomu. Ponieważ szachy są zbyt złożoną grą na ocenienie wszystkich możliwych pozycji, dlatego prace szły w kierunku efektywnego przeszukiwania skróconych drzew gry i eliminacji nieużytecznych kontynuacji. Pojawiła się ocena pozycji, która było początkowo tworzona na podstawie wiedzy ludzi na temat chwilowych korzyści w grze (przewaga materialna, chroniony król itp.). Wzrost szybkości komputerów pozwalał na głębsze przeszukiwanie wariantów, a wzrost ich pamięci na ulokowanie w nich pojemnych bibliotek debiutowych czy końcówkowych.

Działanie algorytmów coraz mniej przypominało podejście ludzi. Pojawiły się elementy heurystyki; czy ktoś rozsądny uzależniłby wybór następnego posunięcia od rzutu monetą? Algortymy szachowe tak nie czyniły jednak analiza losowych kontynuacji gry zaczęła mieć dużą rolę.

W końcu AlphaZero i użycie do jego działania sieci neuronowych dokonało prawdziwej rewolucji: sztuczna inteligencja sama nauczyła się grać w szachy i to z fantastycznym efektem.

Ciemną stroną siły i łatwej dostępności programów szachowych stały się elektroniczne oszustwa w czasie partii, nawet granych na żywo. Dyskwalifikacjami objęci byli nawet arcymistrzowie. W wykrywaniu graczy, korzystających z podpowiedzi programów komputerowych pomagają... te same programy! Okazuje się, że w trakcie gry środkowej, gdy jest największa mnogość wariantów, zgodność posunięć czołowych szachistów z proponowanymi przez najlepsze silniki szachowe raczej nie przekracza 70%. W ten sposób porównując partie szachowe z sugestiami silników szachowych można namierzyć oszustów – na portalu chess.com każdego dnia około 800 kont jest banowanych za oszukiwanie.

Co przyniesie przyszłość w dziedzinie komputerów szachowych? Trudno przewidzieć. Na pewno na rozwiązanie gry szachy się nie zanosi. I bardzo dobrze – niech pozostanie tajemnicą, czy lepiej grać białymi, czy czarnymi. A może przy najlepszej grze obu graczy partia zakończy się remisem? Twierdzenie von Neumanna mówi nam, że dokładnie jeden z tych trzech przypadków musi zajść. Który – tego dziś nikt nie wie.

Literatura

1. P. Auer, N. Cesa-Bianchi, P. Fisher, *Finite-time Analysis of the Multiarmed Bandit Problem*, Machine Learning. 47 (2/3) (2002): 235–256.
2. J. Blüml, J. Czech, K. Kersting, *AlphaZe⋆⋆ : AlphaZero-like baselines for imperfect information games are surprisingly strong*, Frontiers in Artificial Intelligence 6 (2023).

3. S. Maharaj, N. Polson, A. Turk, *Chess Ai: Competing Paradigms for Machine Intelligence*, Entropy, 24(4) (2022), pp. 550.
4. J. von Neumann, *Zur Theorie der Gesellschaftsspiele*, Math. Ann. 100 (1928): 295–320.
5. C. Schannon, *Programming a Computer for Playing Chess*, Philosophical Magazine. 41 (314), 1950.
6. https://maroonchess.com/how-does-stockfish-work/ Dostęp 20.08.2023