Zastosowanie algorytmu UCT do stworzenia sztucznej inteligencji grającej w *Connect4*

Dokumentacja końcowa

Patryk Fijałkowski
Mateusz Burczaniuk

9 czerwca 2020

1 Opis problemu

Monte-Carlo Tree Search to heurystyka, której celem jest podejmowanie decyzji w pewnych zadaniach sztucznej inteligencji, na przykład wybieranie ruchów w grach. Metoda jest oparta na przeszukiwaniu możliwych stanów gry zapisanych w wierzchołkach drzewa i losowym symulowaniu rozgrywek. Algorytmy MCTS opierają się na rozbudowywaniu drzewa ze stanami gry przez iteracyjne wykonywanie czterech faz. Jednym z najpowszechniejszych wariantów MCTS jest algorytm UCT. W niniejszej pracy autorzy zbadają 3 warianty metody UCT: UCB1, UCB-V i UCB-Minimal. Ponadto, zweryfikowane zostaną hipotezy postawione w dokumentacji wstępnej projektu.

1.1 Algorytm UCT

UCT jest wariantem metody MCTS, który stara się zachować równowagę między eksploatacją bardziej obiecujących ruchów a eksploracją tych rzadko odwiedzonych. Formuła, która odpowiada za wyznaczenie najbardziej obiecującego wierzchołka w fazie wyboru MCTS jest przedstawiona jako wyrażenie (1).

$$\frac{w_i}{n_i} + c\sqrt{\frac{\ln N_i}{n_i}} \tag{1}$$

W wyrażeniu (1), indeks i odnosi się do liczby wykonanych przez algorytm iteracji, czyli czterech faz MCTS. W pierwszym składniku sumy wyrażenia (1), licznik w_i oznacza sumę wszystkich wypłat w danym węźle, a mianownik n_i oznacza liczbę rozegranych symulacji. Zatem ułamek ten przyjmuje wartości większe dla ruchów o większej średniej wygranej, co odpowiada ze eksploatację drzewa. Drugi składnik sumy wyrażenia (1) przyjmuje wartości większe dla wierzchołków, dla których wykonano mniej symulacji i odpowiada eksploracji drzewa. $N_i = \sum_i n_i$, a c jest parametrem eksploracji, który może być dostosowany do badanego problemu.

1.1.1 Usprawnienie UCB-Minimal

Przedstawione w [2] usprawnienie UCB-Minimal polega na zastosowaniu wzoru (2) w celu wyboru najbardziej obiecującego wierzchołka w fazie selekcji.

$$\frac{w_i}{n_i} + \frac{C_1}{n_i^{C_2}} \tag{2}$$

W wyrażeniu (2), C_1 jest parametrem balansującym eksploatację i eksplorację drzewa. Z kolei C_2 to parametr, którego zadaniem jest skorygować zmniejszenie wpływu eksploracji, które następuje z kolejnymi iteracjami algorytmu. Dobór parametrów powinien zostać wykonany empirycznie – autorzy [2] sugerują wartości $C_1 = 2.5, C_2 = 1$ jako ogólnie optymalne. Przykładowo, badania autorów [3] wykazały, że najlepiej sprawdzają się wartości $C_1 = 8.4, C_2 = 1.8$.

1.1.2 Usprawnienie UCB-V

Przedstawione w [4] usprawnienie UCB-V opiera swoją skuteczność na wykorzystaniu wariancji wypłat powodując, że algorytm w fazie selekcji wybiera jak najlepsze i jak najmniej zróżnicowane ruchy w kontekście oczekiwanej wypłaty. UCB-V polega na zastosowaniu wzoru (3) w celu wyboru najbardziej obiecującego wierzchołka w fazie selekcji.

$$\frac{w_i}{n_i} + \sqrt{2\frac{\sigma_i^2 \cdot \varepsilon}{n_i}} + c\frac{3 \cdot \varepsilon}{n_i} \tag{3}$$

W wyrażeniu (3), σ_i^2 oznacza wariancję wypłat w danym węźle, a ε to tak zwana *funkcja eksploracji*. Zgodnie z sugestią autorów [4], za *funkcję eksploracji* przyjęta zostanie postać zaprezentowana w (4), gdzie ζ jest parametrem, który powinien zostać dopasowany empirycznie.

$$\varepsilon = \zeta \cdot \ln N_i \tag{4}$$

Autorzy [4] sugerują wartości parametrów $c = 1, \zeta = 1$ jako optymalne dla prezentowanego usprawnienia. Z drugiej strony, w zastosowaniu go do gry *Tron*, autorzy pracy [3] najlepsze efekty zaobserwowali dla wartości parametrów $c = 1.68, \zeta = 0.54$.

2 Sposób przeprowadzenia testów

Każdy z przeprowadzonych testów będzie w formie pełnej rozgrywki *Connect4* pomiędzy dwoma agentami. W celu oszacowania, jak dobre decyzje podejmował agent rozpoczynający rozgrywkę, zdefiniowano funkcję $REWARD$ opisaną równaniem (5). Funkcja jest zależna od liczby ruchów m w danej partii i przyjmuje wartości z zakresu $[-1; 0.8]$. Funkcja nie może przyjąć wartości większych od 0.8 ze względu na przewagę pierwszego gracza spowodowaną faktem, że rozpoczyna on rozgrywkę.

$$REWARD(m) = \begin{cases} 0.8 \cdot (1 - \frac{m-7}{35}) & \text{jeśli agent wygrał} \\ \frac{m-7}{35} - 1 & \text{w p.p.} \end{cases} \tag{5}$$

3 Weryfikacja hipotez

W rozdziałach 3.1 - 3.3 zweryfikowano hipotezy postawione w dokumentacji wstępnej projektu. W rozdziale 3.1 autorzy próbują empirycznie wyznaczyć najbardziej optymalne wartości parametrów dla każdego z trzech wariantów UCT, sugerując się wartościami referencyjnymi zasięgniętymi z innych badań. Rozdział 3.2 jest poświęcony sprawdzeniu wpływu liczby iteracji MCTS na jakość podejmowanych decyzji przez warianty UCT. Finalnie, w rozdziale 3.3 wyznaczono najlepszą konfigurację UCT.

3.1 Optymalne parametry

W celu wyznaczenia najlepszych parametrów dla każdego z trzech analizowanych wariantów UCT, przeprowadzono rozgrywki z algorytmem heurystycznym. Dla każdego wariantu sprawdzono 20 próbnych konfiguracji parametrów, a każdą z konfiguracji sprawdzono dla 15 wartości ziarna generatora liczb losowych, by zmniejszyć wpływ losowości na działanie algorytmu. W każdym teście algorytm MCTS wykonywał 15.000 iteracji. Jako ocenę każdej konfiguracji przyjęto średnią arytmetyczną wartości funkcji $REWARD$ otrzymanych po zakończonych rozgrywkach. Wyniki testów ukazane są w tabelach 1 - 3.

Wartość c	Ocena
2	0.552
1.41	0.529
1.7	0.484
1.6	0.425
1.5	0.415
1.45	0.401
1	0.153
0.09	-0.448
0.01	-0.563
0	-0.702

Tab. 1: Ocena algorytmu UCB1 w zależności od parametru eksploracji

Jak widać w tabeli 1, algorytm UCB1 został najlepiej oceniony dla wartości parametru eksploracji $c = 2$. Wraz ze zwiększaniem i zmniejszaniem wartości parametru, algorytm był oceniany gorzej. Ponadto, w przypadku $c = 0$, kiedy algorytm eksploatował jedynie najbardziej obiecujące ruchy, podejmował najgorsze decyzje. Wartość sugerowana przez autorów algorytmu w [1] ($c = 1.41$) została oceniona nieznacznie gorzej względem $c = 2$.

Wartość c	Wartość ζ	Ocena
1.4	0.5	0.560
2	0.5	0.510
1.68	0.54	0.494
1.7	0.6	0.478
1.5	0.5	0.462
0.9	0.9	0.457
1	1	0.366
1.5	0.4	0.289
120	30	-0.007
0.1	0.05	-0.513

Tab. 2: Ocena algorytmu UCB-V w zależności od parametrów c i ζ

Analizując tabelę 2, wnioskuje się, że algorytm UCB-V działa najlepiej w konfiguracji $(c = 1.4, \zeta = 0.5)$. Wartości sugerowane przez [3] i [4] zostały ocenione gorzej. Podczas testów używano wyłącznie sugerowanej *funkcji eksploracji*, przyjęto $\varepsilon = \zeta \cdot \ln N_i$.

Wartość C_1	Wartość C_2	Ocena
11	1	-0.091
2.5	1	-0.272
2.9	1.4	-0.289
12	5	-0.297
8.4	1.8	-0.349
3	2	-0.366
1.8	8.4	-0.452
3	3	-0.508
26	26	-0.522
9.4	2.8	-0.556

Tab. 3: Ocena algorytmu UCB-Minimal w zależności od parametrów C_1 i C_2

Algorytm UCB-Minimal wypadł najgorzej w porównaniu – nawet najlepiej dobrane wartości parametrów C_1 i C_2 skutkowały ujemnym bilansem zwycięstw i porażek. Zgodnie z tabelą 3, algorytm gra najlepiej w konfiguracji $(C_1 = 11, C_2 = 1)$. Z kolei referencyjne wartości parametrów, zaczerpnięte odpowiednio z [2] i [3], wiążą się z niższą oceną algorytmu.

Algorytm	Ocena
UCBV $(1.4, 0.5)$	0.560
UCB1 (2)	0.552
UCB1 (1.41)	0.529
UCBV $(2, 0.5)$	0.510
UCB1 (1.7)	0.484
UCBV $(1.7, 0.6)$	0.478
UCBV $(1.5, 0.5)$	0.462
UCBV $(0.9, 0.9)$	0.457
UCB1 (3)	0.438
UCBV $(1.1, 1.1)$	0.427

Tab. 4: Ocena najlepszych konfiguracji algorytmów

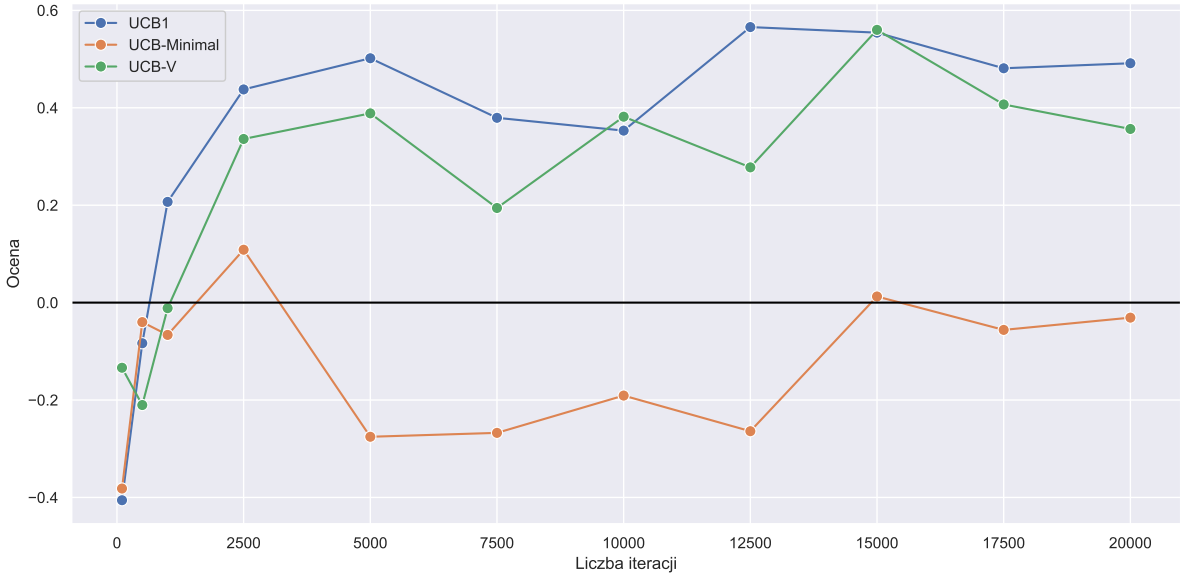
Tabela 4 prezentuje, który wariant UCT został najlepiej oceniony w rozgrywkach z algorytmem heurystycznym. W celu porównania skuteczności każdego z algorytmów, wyznaczono również średnie arytmetyczne wartości funkcji $REWARD$ po wszystkich rozgrywkach. Średnie oceny algorytmów ukazano w tabeli 5.

Algorytm	Średnia ocena
UCB-V	0.306
UCB1	0.112
UCB-Minimal	-0.472

Tab. 5: Średnie oceny algorytmów

3.2 Wpływ iteracji

W celu sprawdzenia, jak liczba iteracji MCTS wpływa na poprawę decyzji algorytmu, przeprowadzono testy dla najbardziej optymalnych konfiguracji parametrów każdego wariantu: UCB1, UCB-V i UCB-Minimal. Oceniono rozgrywki z algorytmem heurystycznym po wykonaniu 100, 500, 1000, 2500, 5000, 7500, 10000, 12500, 15000, 17500 i 20000 iteracji.



Rys. 1: Wpływ liczby iteracji na jakość podejmowanych decyzji

Wyniki analiz zostały zaprezentowane na wykresie na rysunku 1. Ocena każdego z algorytmów osiąga stosunkowo wysokie wartości przy liczbie iteracji 2500, następnie maleje, by przy liczbie iteracji 15000 osiągnąć najwyższą wartość. Wykonanie 2500 iteracji MCTS zdaje się być najbardziej optymalnym rozwiązaniem, równoważącym jakość podejmowanych decyzji i czas obliczeń. Wartym zauważenia jest również fakt, że algorytm UCB-Minimal osiąga relatywnie najlepsze wyniki przy najniższych zakresach iteracji, a przy 2500 iteracji uzyskuje najwyższe osiągane przez siebie oceny.

3.3 Najlepszy wariant

Najlepszy z wariantów zostanie wyłoniony na podstawie rozgrywek *każdy z każdym*. Zostały wybrane 4 najlepsze konfiguracje algorytmów i pomiędzy każdą parą przeprowadzono 80 rozgrywek. Pierwszą połowę z nich rozpoczynał agent korzystający z pierwszej konfiguracji, a drugą – korzystający z drugiej. Wyniki przeprowadzonych testów ukazano w tabeli 6, 7 i 8.

	UCB-V $(1.4, 0.5)$	UCB1 (2)	UCB1 (1.41)	UCB-V $(2, 0.5)$
UCBV $(2, 0.5)$	0.1316	-0.0597	0.0701	
UCB1 (1.41)	-0.0336	-0.1616		
UCB1 (2)	0.1611			
UCB-V $(1.4, 0.5)$				

Tab. 6: Ocena najlepszych konfiguracji algorytmów

Algorytm	Ocena
UCB1 (2)	0.3824
UCB-V $(2, 0.5)$	0.142
UCB-V $(1.4, 0.5)$	-0.2591
UCB1 (1.41)	-0.2653

Tab. 7: Sumaryczne oceny algorytmów

Zgodnie z wynikami prezentowanymi w tabeli 6, wariant algorytmu UCB1 ($c = 2$) wygrał w ocenach ze wszystkimi pozostałymi wariantami i w tabeli 7 widać, że uzyskał również najwyższą sumaryczną ocenę. Wariant UCB-V ($c = 2, \zeta = 0.5$) zajął w rankingu drugie miejsce, przegrywając nieznacznie z wariantem UCB1 ($c = 2$). Warto zaznaczyć, że zgodnie z tabelą 8, wariant UCB-V ($c = 2, \zeta = 0.5$) doprowadził do największego odsetku remisów (13.4%) podczas rozgrywek, co znacząco wpłynęło na pogorszenie jego oceny.

	UCB-V $(1.4, 0.5)$	UCB1 (2)	UCB1 (1.41)	UCB-V $(2, 0.5)$
UCB-V $(2, 0.5)$	7.5%	16.25%	16.25%	
UCB1 (1.41)	12.5%	5%		
UCB1 (2)	8.75%			
UCB-V $(1.4, 0.5)$				

Tab. 8: Odsetek remisów w przeprowadzonych rozgrywkach

Literatura

[1] Levente Kocsis, Csaba Szepesvári, *Bandit based Monte-Carlo Planning*, European Conference on Machine Learning, Berlin, Germany, September 18–22, 2006.

[2] Francis Maes, Louis Wehenkel, Damien Ernst, *Automatic Discovery of Ranking Formulas for Playing with Multi-armed Bandits*, European Workshop on Reinforcement Learning, Athens, Greece, September 9–11, 2011.

[3] Pierre Perick, David L. St-Pierre, Francis Maes, Damien Ernst, *Comparison of Different Selection Strategies in Monte-Carlo Tree Search for the Game of Tron*, IEEE Conference on Computational Intelligence and Games, Granada, Spain, September 12–15, 2012.

[4] Jean-Yves Audibert, Remi Munos, Csaba Szepesvári, *Tuning Bandit Algorithms in Stochastic Environments*, Algorithmic Learning Theory 18th International Conference, Sendai, Japan, October 1–4, 2007.