Wprowadzenie do sztucznej inteligencji - ćwiczenie nr 3

Jan Górski

8 kwietnia 2024

Spis treści

1 Zadanie

1.1 Zadanie

Zaimplementować algorytm min-max z przycinaniem alfa-beta. Algorytm ten należy zastosować do gry w proste warcaby (checekers/draughts). Niech funkcja oceny planszy zwraca różnicę pomiędzy stanem planszy gracza a stanem przeciwnika. Za pion przyznajemy 1 punkt, za damkę 10 p. Proszę nie zapomnieć o znacznej nagrodzie za zwycięstwo.

Przygotowałem dla Państwa kod, który powinien ułatwić wykonanie zadania. Nie można używać kodu z Internetu, czy bardziej ogólnie, kodu, którego nie jest się autorem.

Wiem co jest dostępne w Internecie, większość dostępnych implementacji ma cechy szczególne, po których łatwo je rozpoznać.

Zasady gry (w skrócie: wszyscy ruszają się po 1 polu. Pionki tylko w kierunku wroga, damki w dowolnym) z następującymi modyfikacjami:

- Bicie nie jest wymagane.
- Dozwolone jest tylko pojedyncze bicie (bez serii).

1.2 Pytania

- Czy gracz sterowany przez AI zachowuje się rozsądnie z ludzkiego punktu widzenia? Jeśli nie to co jest nie tak?

Niech komputer gra z komputerem (bez wizualizacji), zmieniamy parametry jednego z oponentów, badamy jak zmiany te wpłyną na liczbę jego wygranych. Należy zbadać wpływ:

- Głębokości drzewa przeszukiwań
- Alternatywnych funkcji oceny stanu (nadal ocena jest różnicą pomiędzy oceną stanu gracza a oceną przeciwnika), np.:
    1. nagrody jak w wersji podstawowej + nagroda za stopień zwartości grupy (jest dobrze jak wszyscy są blisko siebie lub przy krawędzi planszy)
    2. za każdy pion na własnej połowie planszy otrzymuje się 5 nagrody, na połowie przeciwnika 7, a za każdą damkę 10.
    3. za każdy nasz pion otrzymuje się nagrodę w wysokości: (5 + numer wiersza, na którym stoi pion) (im jest bliżej wroga tym lepiej), a za każdą damkę dodatkowe 10.

2 Część teoretyczna

2.1 Gry dwuosobowe

Znalezienie optymalnej strategii w grze można sprowadzić do problemu przeszukiwania przestrzeni stanów. Przykładem takiej gry może być skończona gra dwuosobowa o sumie zerowej. W takiej sytuacji zysk jednego gracza jest równy stracie drugiego gracza - interesy graczy są przeciwstawne. Takie gry możemy podzielić na kategorie pod względem dostępu do informacji (informacja o stanie planszy może być pełna lub niepełna dla graczy) lub pod względem tego, czy w grze występuje czynnik losowy (mamy wtedy podział na gry deterministyczne i hazardowe).

Poniższe ćwiczenie dotyczy gry warcaby (ang. checkers), która jest grą deterministyczną, w której obaj gracze mają dostęp do pełnej informacji o stanie gry i planszy.

Grę można opisać jak $\langle S, P, s_0, T, w \rangle$, gdzie:

- $s \in S$ - stan (np. ustawienie figur na szachownicy) i informacja, kto wykonuje ruch,
- $p \in P$ - funkcja następnika, reprezentuje posinięcia w grze, $p : S \leftarrow S$ pokazu dostępne stany potomne dla danego stanu,
- $s_0 \in S$ - stan początkowy,
- $T \subseteq S$ - zbiór stanów terminalnych (kończących grę),
- w - funkcja wypłaty zdefiniowana dla stanów terminalnych $s \in T$.

2.2 MiniMax

Algorytm MiniMax podaje ocenę dla danego węzła, przeglądając węzły potomne na określoną głębokość. Wykorzystuje on funkcję heurystyczną $h(s)$, która ocenia stan gry dostarczając ocenę lub jej oszacowanie.

W zadaniu badano różne dostarczone funkcje heurystyczne oceniające stan planszy.

W zadaniu należało wykorzystać algorytm MiniMax z przycinaniem alfa - beta. Polega ono na uproszczeniu drzewa przeszukiwań poprzez nieprzeglądanie ścieżek, których wybór byłby gorszy od obecnie najlpszego wyboru dla innej ścieżki.

2.3 Algorytm MiniMax z przycinaniem α - β: pseudokod

Algorithm 1: AlfaBeta

Data: s : stan i informacja, kto wykonuje ruch,
d : głębokość,
$move\_max$: zmienna binarna, czy rusza się Max czy Min,
α : najlepsza obecna wypłata dla Max.
Przy początkowym wywołaniu równa $-\infty$.,
β : najlepsza obecnie wypłata dla gracza Min.
Przy początkowym wywołaniu równa ∞.

Result: α lub β:
wartość funkcji oceny dla najlepszego rozwiązania przy zadanych parametrach

```
1  begin
2      if s ∈ T or d = 0 then
3          return h(s)
4      end if
5      U ← P(s)
6      if move_max then
7          for u ∈ U do
8              α ← max{α, AlfaBeta(u, d − 1, not move_max, α, β)}
9              if α ≤ β then
10                 return α
11             end if
12         end for
13         return α
14     else
15         for u ∈ U do
16             α ← min{α, AlfaBeta(u, d − 1, not move_max, α, β)}
17             if α ≤ β then
18                 return β
19             end if
20         end for
21         return β
22     end if
23 end
```

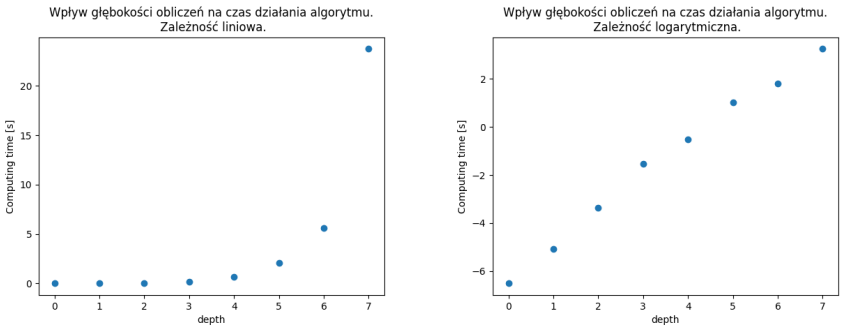
3 Część praktyczna

3.1 Subiektywna ocena działania algorytmu

Celem pierwszej części eksperymetu było zaimplementowanie algorytmu dla gry w warcaby i subiektywna ocena wybranych przez algorytm posunięć.

Algorytm działał zgodnie z przyjętą heurystyką: różnica sumy wartości własnych pionków z sumą wartości pionków przeciwnika. Dla głębokości równej 1 zachowanie komputera można nazwać zachłannym- Gdy tylko mógł wykonać bicie, to to robił. Jednak wielokrotnie samemu poruszał się na pole, z którego mógł zostać zbity bez możliwości odbicia.

Dla kontrastu, przy głębokości równej 6 jego gra była znacznie lepsza. Ruchy komputera wydawały się być rozsądne. Niezależnie jednak od tego, czy ruch był „naturalnym" odbiciem jako reakcja na bicie, czy też występował w trudniejszej do ecenienia pozycji, czas potrzebny na wykonanie ruchu był tak samo długi.



(a) Wykres liniowy.

(b) Wykres logarytmiczny.

Rysunek 1: Wykresy złożoności czasowej działania algorytmu w zależności od głębokości przeszukiwania.

3.2 Uwagi dotyczące implementacji algorytmu

- Kilkakrotnie podczas testów zaobserwowałem, że w pewnym momencie obaj gracze sterowani przez algorytm zaczynały powtarzać posunięcia, co uniemożliwiało określenie, który z nich wygrał. Dodatkowo, wywołując algorytm dla takich samych parametrów otrzymywałem taki sam przebieg partii. Żeby zmienić te rzeczy, mając listę par <ruch, ocena> przed posortowaniem ich ze względu na wartość oceny mieszałem je i dopiero wtedy sortowałem. Pozwalało to na wybór w tej samej sytuacji innych ścieżek, co zmniejszało monotonny charakter rozgrywki.
- Głębokość równa 0 została zaimplementowana jako wybór losowego ruchu spośród dostępnych.
- W przypadku heurystyki stosującej ocenę zwartości grupy lub bycie przy krawędziach planszy zastosowano następującą interpretację: „zwartość" była wyznaczana zarówno poziomo

jak i poziomo. Wyliczano ją jako odchylenie standardowe liczone dla zbioru położeń (wybranej składowej) wszystkich figur danego koloru. Następnie obie sumy (dla poziomu i pionu) sumowano. Zwracano różnicę oceny pomiędzy oceną białego a czarnego przeskalowaną o współczynnik. Współczynnik wybrano po obserwacji działania algorytmu dla różnych wartości. Obecność figury przy krawędzi miało wpływ na ocenę danego piona rzędu 0.1.

3.3 Badanie wpływu głębokości drzewa przeszukiwań oraz rodzaju funkcji oceny stanu na liczbę wygranych algorytmu

Celem drugiej części eksperymetu było zbadanie zachowania komputera dla różnych wartości głębokości drzewa przeszukiwań oraz dla różnych funkcji heurystycznych, które miały zostać zaimplementowane.

Wpływ głębokości drzewa przeszukiwań oraz rodzaju funkcji oceny stanu na liczbę wygranych algorytmu. Badany gracz niebieski.		
heurystyka (biały | niebieski)	głębokość (biały | niebieski)	wynik (z perspektywy białego) (wygrana | remis | przegrana)
(default | default)	(2 | 1)	(6 | 1 | 13)
	(2 | 2)	(0 | 0 | 20)
	(2 | 3)	(0 | 0 | 20)
	(2 | 4)	(1 | 0 | 19)
	(2 | 5)	(0 | 0 | 20)
(default | tight)	(2 | 1)	(13 | 6 | 1)
	(2 | 2)	(5 | 10 | 5)
	(2 | 3)	(4 | 12 | 4)
	(2 | 4)	(3 | 17 | 0)
	(2 | 5)	(7 | 9 | 4)
(default | enemy side)	(2 | 1)	(7 | 0 | 13)
	(2 | 2)	(0 | 0 | 20)
	(2 | 3)	(0 | 0 | 20)
	(2 | 4)	(0 | 0 | 20)
	(2 | 5)	(0 | 0 | 20)
(default | hight)	(2 | 1)	(9 | 1 | 10)
	(2 | 2)	(0 | 0 | 20)
	(2 | 3)	(0 | 0 | 20)
	(2 | 4)	(0 | 0 | 20)
	(2 | 5)	(0 | 0 | 20)

Tablica 1: Rezultaty pomiarów. Badany gracz niebieski. Parametry gracza białego: głębokość - 2, heurystyka - domyślna.

Dla domyślnej heurystyki gdy głębokość przeszukiwań niebieskiego wynosiła więcej niż 1, gracz niebieski niemal zawsze wygrywał.

Zastosowanie heurystyki „tight” skutkowało zwiększaniem się liczby remisów.

Pozostałe heurystyki dawały podobne rezultaty do domyślnej.

Wpływ głębokości drzewa przeszukiwań oraz rodzaju funkcji oceny stanu na liczbę wygranych algorytmu. Badany gracz: biały.		
heurystyka (biały | niebieski)	głębokość (biały | niebieski)	wynik (z perspektywy białego) (wygrana | remis | przegrana)
(default | default)	(1 | 2)	(0 | 0 | 20)
	(2 | 2)	(0 | 0 | 20)
	(3 | 2)	(0 | 0 | 20)
	(4 | 2)	(0 | 0 | 20)
	(5 | 2)	(0 | 0 | 20)
(tight | default)	(1 | 2)	(0 | 7 | 13)
	(2 | 2)	(0 | 3 | 17)
	(3 | 2)	(4 | 12 | 4)
	(4 | 2)	(0 | 0 | 20)
	(5 | 2)	(0 | 0 | 20)
(enemy side | default)	(1 | 2)	(0 | 0 | 20)
	(2 | 2)	(1 | 1 | 18)
	(3 | 2)	(0 | 0 | 20)
	(4 | 2)	(1 | 1 | 18)
	(5 | 2)	(0 | 0 | 20)
(hight | default)	(1 | 2)	(0 | 0 | 20)
	(2 | 2)	(0 | 0 | 20)
	(3 | 2)	(0 | 0 | 20)
	(4 | 2)	(0 | 0 | 20)
	(5 | 2)	(0 | 0 | 20)

Tablica 2: Rezultaty pomiarów. Badany gracz biały. Parametry graczaj niebieskiego: głębokość - 2, heurystyka - domyślna.

Pomimo tego, że głębokość białego zmieniała się od wartości 1 do 5, podczas gdy głębokość niebieskiego pozostawała stała równa 2, dla każdej heurystyki poza heurystyką „tight", to niebieski wygrywał większość partii. Przeprowadzenie badania dla większej wartości głębokości białego nie było możliwe z uwagi na czas potrzebny na wykonanie testów.

W związku z powyższym przeprowadzono jeszcze jeden test. W stosunku do poprzedniego zmieniła się w nim tylko jedna rzecz: głębokość niebieskiego została zmieniona na wartość 1.

Wpływ głębokości drzewa przeszukiwań oraz rodzaju funkcji oceny stanu na liczbę wygranych algorytmu. Badany gracz: biały.		
heurystyka (biały | niebieski)	głębokość (biały | niebieski)	wynik (z perspektywy białego) (wygrana | remis | przegrana)
(default | default)	(1 | 2)	(9 | 0 | 11)
	(2 | 2)	(7 | 0 | 13)
	(3 | 2)	(9 | 0 | 11)
	(4 | 2)	(9 | 0 | 11)
	(5 | 2)	(11 | 0 | 9)
(tight | default)	(1 | 2)	(0 | 5 | 15)
	(2 | 2)	(0 | 1 | 19)
	(3 | 2)	(0 | 0 | 20)
	(4 | 2)	(0 | 0 | 20)
	(5 | 2)	(0 | 0 | 20)
(enemy side | default)	(1 | 2)	(10 | 0 | 10)
	(2 | 2)	(14 | 0 | 6)
	(3 | 2)	(8 | 0 | 12)
	(4 | 2)	(16 | 0 | 4)
	(5 | 2)	(9 | 1 | 10)
(hight | default)	(1 | 2)	(10 | 0 | 10)
	(2 | 2)	(8 | 0 | 12)
	(3 | 2)	(9 | 0 | 11)
	(4 | 2)	(6 | 0 | 14)
	(5 | 2)	(12 | 1 | 7)

Tablica 3: Rezultaty pomiarów. Badany gracz biały. Parametry graczaj niebieskiego: głębokość - 1, heurystyka - domyślna.

Zmniejszenie wartości głębokości niebieskiego spowodowało, że biały częściej wygrywał dla każdej heurystyki innej od „tight".

Dla pozostałych heurystyk nie widać wpływu głębokości działania algorytmu na rezultaty białego. Niezależnie od przyjętej głębi biały wygrywa około połowę partii pomimo tego, że w niektórych przypadkach głebokość jego obliczeń jest większa nawet o 4 w porównaniu do niebieskiego gracza.

4 Wnioski

Udało się zaimplementować minimax z przycinaniem $\alpha - \beta$.

Głębokość przeszukiwania drzewa znaczącą wpływa na na czas obliczeń.

Z wyników partii nie można jednoznacznie określić, która funkcja heurystyczna dawała najlepsze rezultaty. Wyniki partii w większości nie wyglądały, jakby były skorelowane z głębokością przeszukiwań. Jedynie w sytuacji, gdy głębokość niebieskiego wynosiła 1 przy badaniu jego działania, wynik różnił się od pozostałych. Funkcja heurystyczna „tight" dawała najgorsze rezultaty, w porównaniu do reszty heurystyk dla niebieskiego. Dla białego pozwalała jedynie trochę częściej uzyskać remis, gdy reszta heurystyk powodowała niemal wyłącznie porażki.

Przyczyną dziwnych rezultatów badań może być charakter gry do której algorytm został zastosowany. Gracz niebieski odpowiada na ruchy gracza białego, co być może jest preferowaną opcją, ponieważ czeka się wtedy tylko na błąd oponenta. Ustawienie figur na szachownicy wymusza to, że w końcu dojdzie do kontaktu i serii wymian figur. Czarny mając ostateczny ruch może na tym zyskać. Warto zaznaczyć, że idealna gra warcab kończy się remisem, więc biały może mieć tutaj mniejszą korzyść z zaczynania, niż ma to miejsce w przypadku niebieskiego.

W celu lepszego zbadania algorytmu należałoby przeprowadzić testy dla większych wartości głębokości przeszukiwania drzewa. Dodatkowo możnaby spróbować innych heurystyk lub zmodyfikować te już użyte.