Sprawozdanie

Game Solving Algorithms

Sztuczna inteligencja i inżynieria wiedzy
Tomasz Mosur

Zasady gry

To gra strategiczna zwana niegdyś „Pogrzebem”. Podobno grywali w nią porywczy bywalcy knajp portowych. Jest naprawdę emocjonująca. Pole gry to kwadrat 7 × 7 lub 8 × 8 kratek. Gracze powinni używać pisaków w dwóch różnych kolorach. Rozpoczynający zamalowuje jedną dowolną kratkę, drugi robi to samo. Jeśli zamalowane kratki wypełnią rząd, kolumnę lub linię ukośną (od krawędzi do krawędzi), otrzymuje się tyle punktów, ile kratek liczy zakolorowany przez nas odcinek (minimum 2 kratki). Zdarzyć się może, że zakolorowanie jednej kratki „zamyka” nawet kilka punktonośnych odcinków, wówczas liczy się każdy odcinek osobno, sumuje i zalicza wszystkie punkty.

Algorytm min-max

Posiadając funkcję **S** oceniającą wartość stanu gry w dowolnym momencie(gracz min chce ten stan zminimalizować, a gracz max zmaksymalizować) obliczamy drzewo wszystkich możliwych stanów w grze do pewnej głębokości(ograniczonej zazwyczaj przez naszą moc obliczeniową). Zakładając, że rozgałęzienie drzewa stanów jest stałe i wynosi **b**(czyli na każdy ruch można odpowiedzieć **b** innymi) , a głębokość **d**(tyle ruchów do przodu symulujemy algorytmem minmax) to mamy stanów końcowych dla których obliczamy wartość stanu gry funkcją **S**. Zaczynamy przeglądanie od stanów końcowych, symulując optymalne wybory dla obu graczy tak aby na głębokości **d**(w liściach drzewa) była dla nich optymalna liczba **S(**stan gry po wykonaniu **d** ruchów). Tak więc gracz min zawsze wybiera ruch który prowadzi do mniejszej wartości końcowej, a gracz max - przeciwnie. Po przeprowadzeniu tej symulacji gracz, który znajduje się w korzeniu drzewa(aktualnie wykonujący ruch) ma pewność, że jego ruch jest optymalny w kontekście informacji o stanie gry z przeprowadzonej symulacji algorytmem minimax na głębokość **d**(tzn. maksymalizuje minimalny zysk).

Pseudokod algorytmu:

```
01 function minimax(node, depth, maximizingPlayer)
02     if depth = 0 or node is a terminal node
03         return the heuristic value of node
04     if maximizingPlayer
05         bestValue := −∞
06         for each child of node
07             v := minimax(child, depth − 1, FALSE)
08             bestValue := max(bestValue, v)
09         return bestValue
10     else    (* minimizing player *)
11         bestValue := +∞
12         for each child of node
13             v := minimax(child, depth − 1, TRUE)
14             bestValue := min(bestValue, v)
15         return bestValue
```

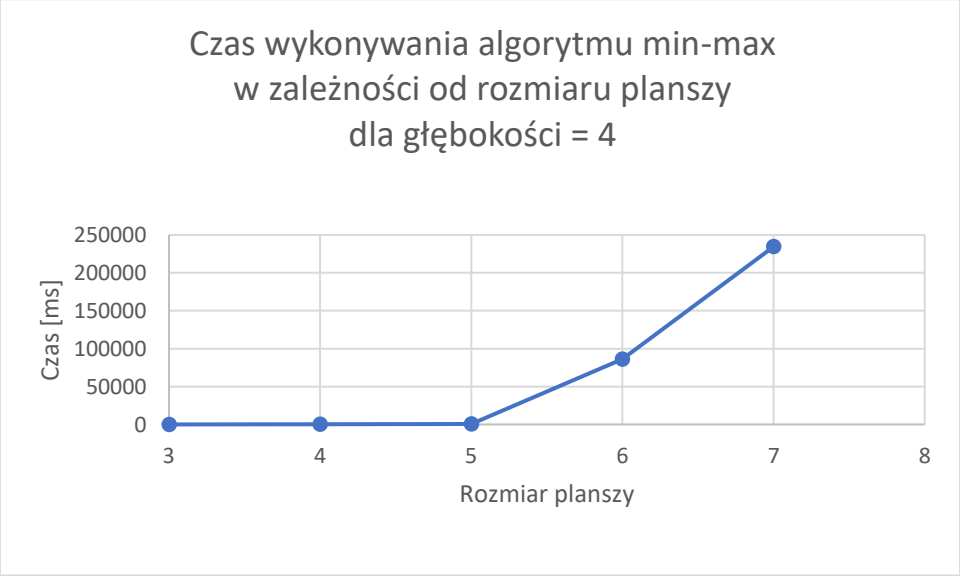
Algorytm min-max z cięciami alfa-beta

Korzyść płynąca z algorytmu alfa-beta leży w fakcie, że niektóre gałęzie drzewa przeszukiwania mogą zostać odcięte. Czas przeszukiwania ograniczony zostaje do przeszukania najbardziej obiecujących poddrzew, w związku z czym możemy zejść głębiej w tym samym czasie. Tak samo jak klasyczny min-max, algorytm należy do algorytmów wykorzystujących metody podziału i ograniczeń (*branch and bound*). Współczynnik rozgałęzienia jest dwukrotnie mniejszy niż w metodzie min-max. Algorytm staje się wydajniejszy, gdy węzły rozwiązywane są układane w porządku optymalnym lub jemu bliskim.

Pseudokod algorytmu:

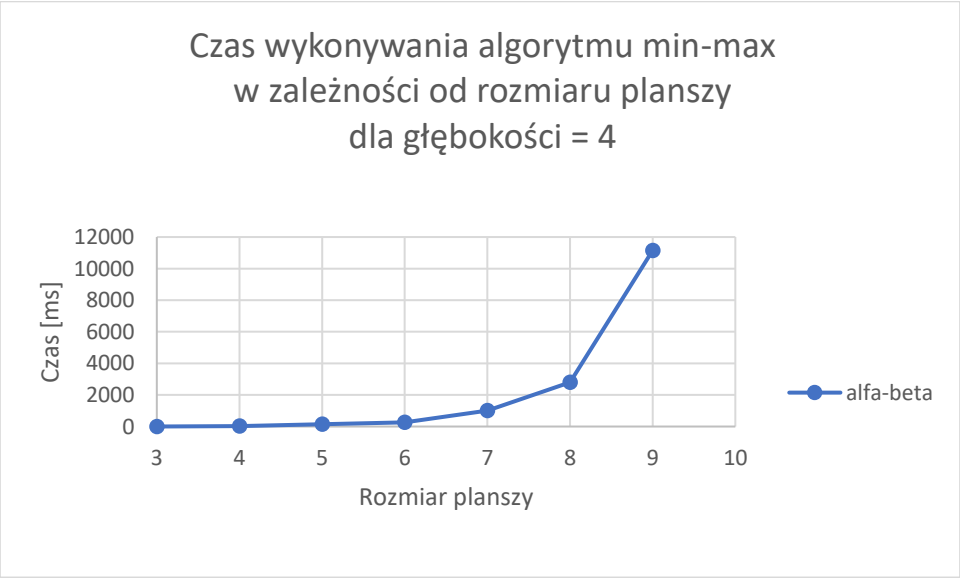
```
funkcja minimax(węzeł, głębokość)
    zwróć alfabeta(węzeł, głębokość, -∞, +∞)

funkcja alfabeta(węzeł, głębokość, α, β)
    jeżeli węzeł jest końcowy lub głębokość = 0
        zwróć wartość heurystyczną węzła
    jeżeli przeciwnik ma zagrać w węźle
        dla każdego potomka węzła
            β := min(β, alfabeta(potomek, głębokość-1, α, β))
            jeżeli α≥β
                przerwij przeszukiwanie  {odcinamy gałąź Alfa}
        zwróć β
    w przeciwnym przypadku {my mamy zagrać w węźle}
        dla każdego potomka węzła
            α := max(α, alfabeta(potomek, głębokość-1, α, β))
            jeżeli α≥β
                przerwij przeszukiwanie  {odcinamy gałąź Beta}
        zwróć α
```

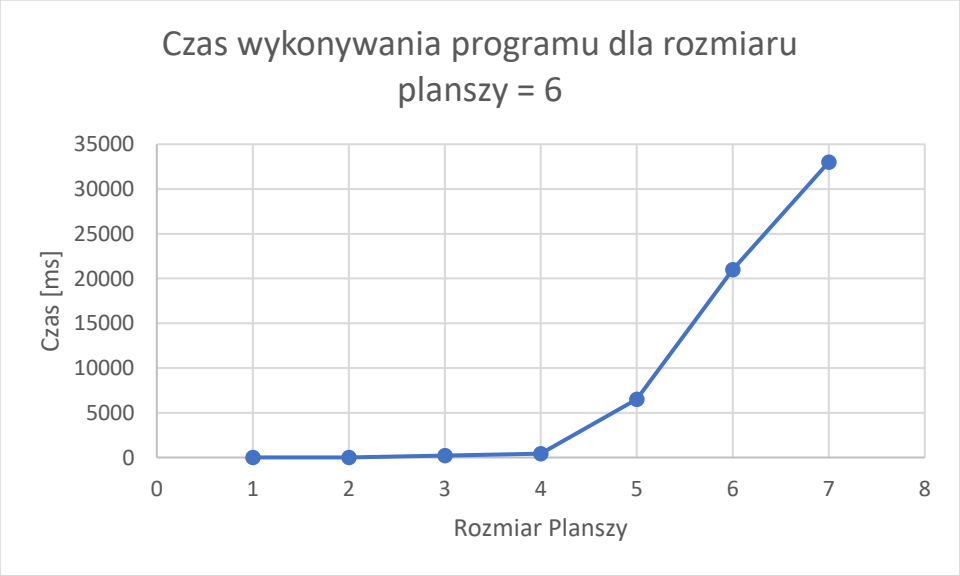
Badanie działania programu:



Widzimy, że czas wykonywania programu jest funkcją wykładniczą. Oznacza to, że przy dużych wartościach planszy będziemy musieli znacznie ograniczyć głębokość drzewa.

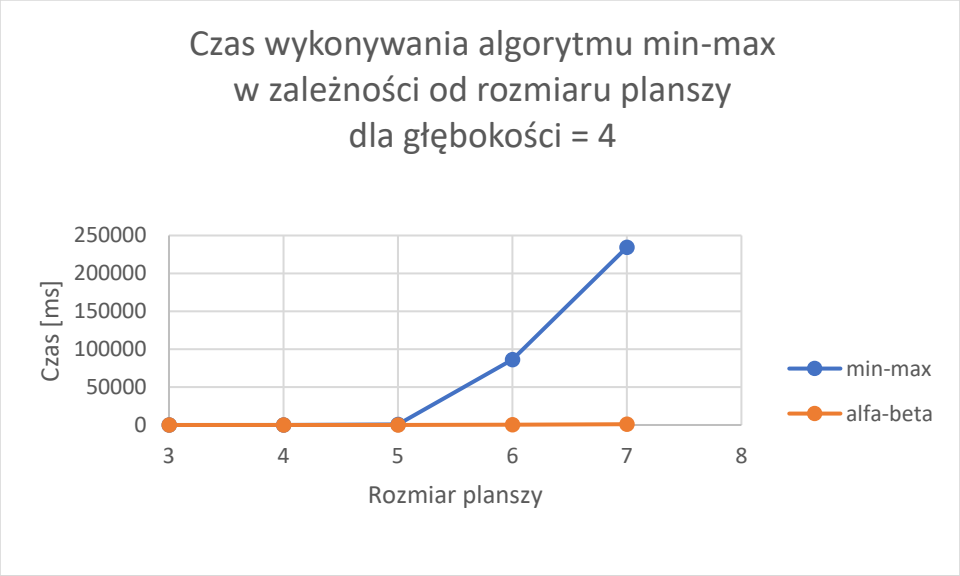
Wpływ algorytmu min-max z cięciami alfa-beta





Czas wykonywania programu rośnie wykładniczo wraz z rozmiarem planszy oraz z głębokością drzewa gry.

Porównanie czasu wykonywania obu algorytmów



Wpływ algorytmu alfa-beta

Stosując algorytm alfa-beta cięć, dzięki odcinaniu poddrzew, które na pewno nie będą rozwiązaniem, jesteśmy w stanie zwiększyć głębokość drzewa gry przy zachowaniu takiego samego czasu rozgrywki.

Dla algorytmu alfa-beta przy rozmiarze planszy 7 byliśmy w stanie osiągnąć głębokość o dwa większą, wynoszącą 6 niż w prostym algorytmie min-max przy zachowaniu takiego samego czasu rozgrywki.

Heurystyki oceny stanu gry

Największa liczba punktów

Funkcja heurystyczna zwraca ilość punktów, jaką posiada gracz w aktualnym stanie gry

Największa różnica punktów

Funkcja heurystyczna zwraca różnicę punktów, jaką posiada gracz w aktualnym stanie gry nad przeciwnikiem

Porównanie heurystyk

Maksymalny czas na ruch: 3s.

DEPTH 4

ROZMIAR PLANSZY	Max-Diff as PLAYER 1	Max-Diff as PLAYER 2	TOTAL Difference
4	-18	8	-10
5	32	-2	30
6	-28	30	2
7	48	-20	28

8	-54	56	2
		Difference	52

DEPTH 5

ROZMIAR PLANSZY	Max-Diff PLAYER 1	Max-Diff PLAYER 2	TOTAL Difference
3	12	-8	4
4	-14	-2	-16
5	18	-2	16
6	-18	30	12
7	26	-20	6
8	-20	56	36
		Difference	54

Można zauważyć, że heurystyka największej różnicy punktów średnio sprawuje się lepiej na niezależnie od głębokości drzewa.

Heurystyki przeszukiwania kolejnych węzłów

TAKE FIRST

Algorytm zawsze schodzi najpierw od rodzica do dzieci znajdujących się po lewej stronie

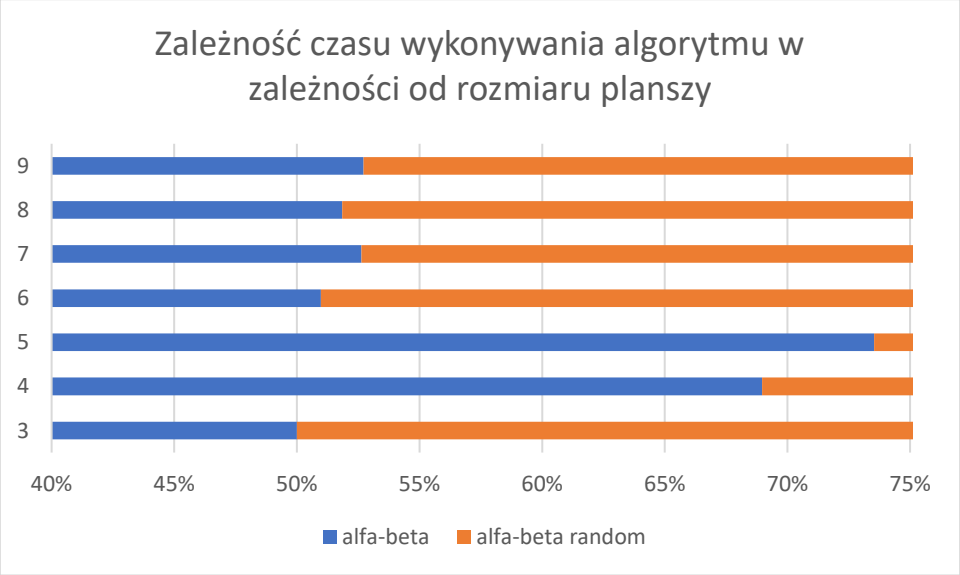
TAKE RANDOM

Algotytm wybiera losowo dziecko rodzica.

Badania przeprowadzono z ustawieniami:

Maksymalny czas ruchu: 2 sekundy.

Głębokość drzewa: 5.



Algorytm wybierający dzieci losowo jest średnio kilka procent szybszy od algorytmu biorącego pierwsze dziecko z kolei.

Dodatkowo, pojedynek człowieka z graczem komputerowym stosującym heurystykę wyboru losowego dziecka jest ciekawszy dla gracza, ponieważ nie priorytetyzuje on pól znajdujących się w lewym górnym rogu

depth 5

Rozmiar planszy	RANDOM as PLAYER 1	RANDOM as PLAYER 2	TOTAL Difference
3	12	-8	4
4	-16	14	-2
5	12	-18	-6
6	-16	18	2
7	30	-34	-4
		Difference	-10

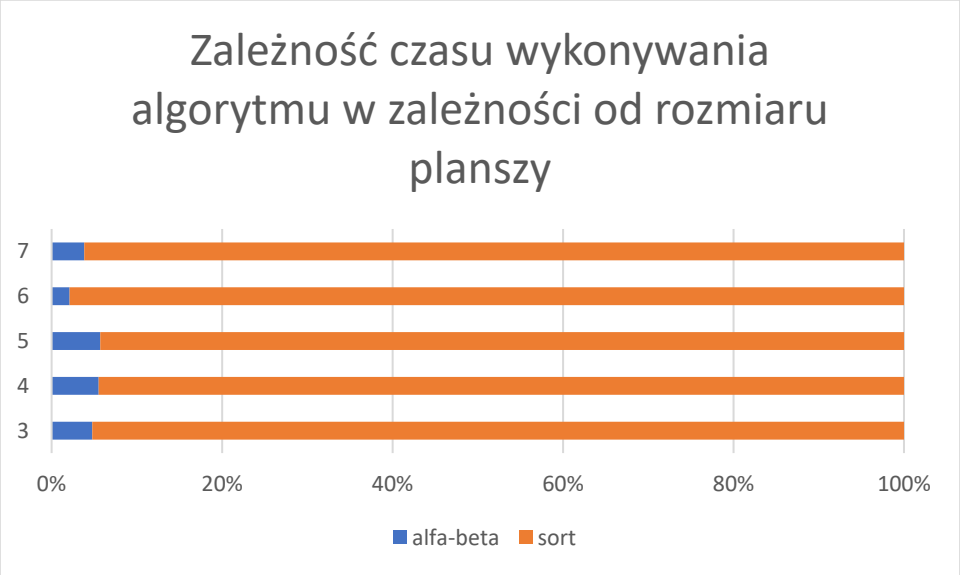
depth 3

Rozmiar planszy	RANDOM as PLAYER 1	RANDOM as PLAYER 2	TOTAL Difference
3	12	-12	0
4	-14	10	-4
5	20	-14	6
6	-18	10	-8
7	16	-20	-4
		Difference	-10

Heurystyka wyboru losowego dziecka w poszczególnych przypadkach potrafi być dużo gorsza lub dużo lepsza od heurystyki wyboru pierwszego dziecka. Jednak w ogólnym rozrachunku wypada gorzej. Może to być spowodowane wpływem pozycjonowania. Koncentracja w jednym rogu planszy może mieć pozytywny wpływ na wynik.

Jeżeli zaczyna grasz korzystający z heurystyki wyboru pierwszego dziecka, losowość wyboru drugiego gracza przestaje mieć znaczenie i każda partia rozgrywana jest w ten sam sposób.

TAKE BEST CHILD



Czas wykonywania programu jest zdecydowanie dłuższy niż czas wykonywania programu z użyciem heurystyki TAKE FIRST. Spowodowane jest to skomplikowaniem operacji sortowania, która nie przynosi aż tak wymiernych rezultatów, ponieważ często wartości węzłów równe są zero. Dopiero pod koniec gry dochodzi do zamykania linii i wartości węzłów zaczynają wypełniać się wartościami