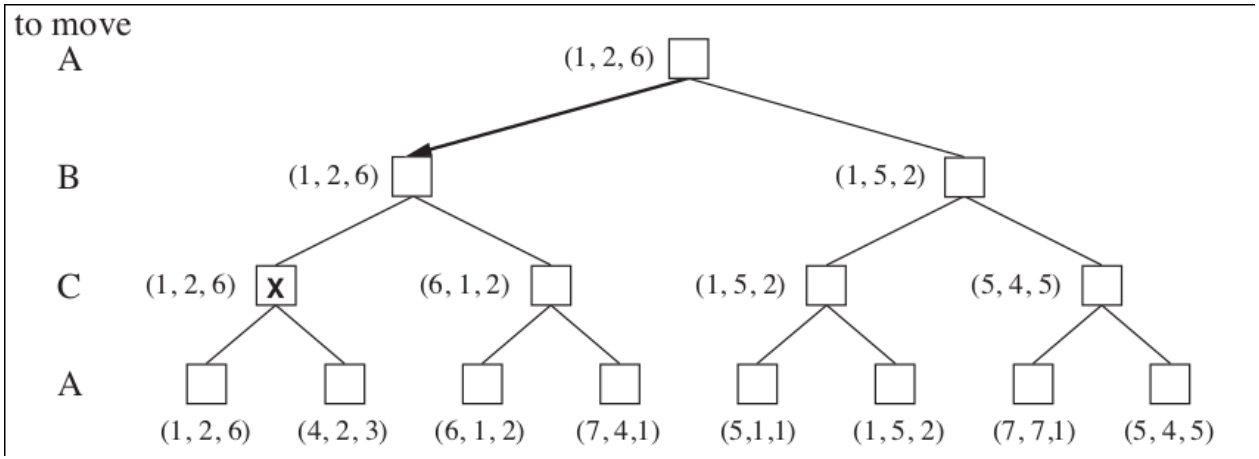
Sztuczna inteligencja. Gry – przyśpieszanie, uczenie i inne zagadnienia

Paweł Rychlikowski

Instytut Informatyki UWr

21 kwietnia 2018

Gry z większą liczbą uczestników



Gry wieloosobowe. Problemy

- Strategia maksymalizująca korzyść pojedynczego gracza w oczywisty sposób nieoptymalna (A mógłby się dogadać z B).
- Kwestie sojuszów, zrywania sojuszów, budowania wiarygodności.
- Czasem używa się: paranoidalnego założenia – gra wieloosobowa staje się jednoosobową, w której oni wszyscy chcą mi zaszkodzić.

Księga otwarć

Uwaga

Początki gier są podobne (bo rozpoczynamy z tego samego stanu startowego)

Z tego wynika, że:

- Możemy np. poświęcić parę godzin, na obliczenie najlepszej odpowiedzi na każdy ruch otwierający.
- Możemy „rozwinąć" początkowy kawałek drzewa (od któregoś momentu tylko dobre odpowiedzi oponenta)
- Możemy skorzystać z literatury dotyczącej początków gry (obrona sycylijska, partia katalońska, obrona bałtycka, i wiele innych)

Spamiętywanie

- Stany mogą się powtarzać (również z zeszłej partii naszego programu).
- Jeżeli mamy oceniony stan z głębokością 6 i dochodzimy do niego z głębokością 3, to opłaca się wziąć tę bardziej prezycyjną ocenę (w dodatku bez żadnych obliczeń).

Uwaga

Potrzebny nam jest efektywny sposób pamiętania sytuacji na planszy.

Tabele transpozycji

- Zapamiętywanie pozycji powinno być efektywne pamięciowo i czasowo.
- Używa się następującego schematu kodowania (Zobrist hashing:
  - Mamy zdania typu: biały goniec na g6 (WB-G6), czarny król na b4 (BK-B4), itd (12×64)
  - Każde z nich dostaje losowy ciąg bitów (popularny wybór: 64 bity)
  - Planszę kodujemy jako **xor** wszystkich prawdziwych zdań o tej planszy.
  - Zauważmy, jak łatwo przekształca się te kody:
    stary-kod = stary-kod **xor** wk-a4 **xor** wk-b5
    to ruch białego króla z a4 na b5

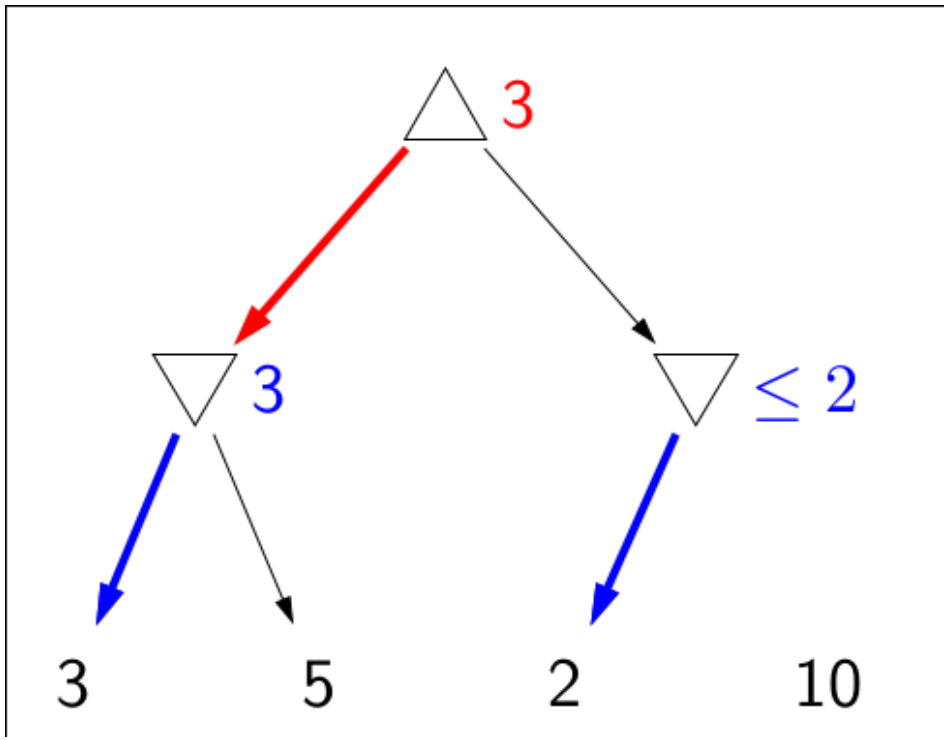
Uwaga

Często nie przejmujemy się konfliktami, uznając że nie wpływają w znaczący sposób na rozgrywkę.

Algorytm Alpha-Beta Search

- Idea 1: nie zawsze musimy przeglądać całe drzewo, żeby wybrać optymalną ścieżkę.
- Idea 2: mamy dwa ruchy
  1. Pierwszy ma wartość z przedziału [2,**5**]
  2. Drugi ma wartość z przedziału [**5**,100]
- Nie musimy ustalać dokładnej wartości pierwszego ruchu, by stwierdzić, że drugi jest lepszy dla Maxa!

Obcinanie fragmentów drzew



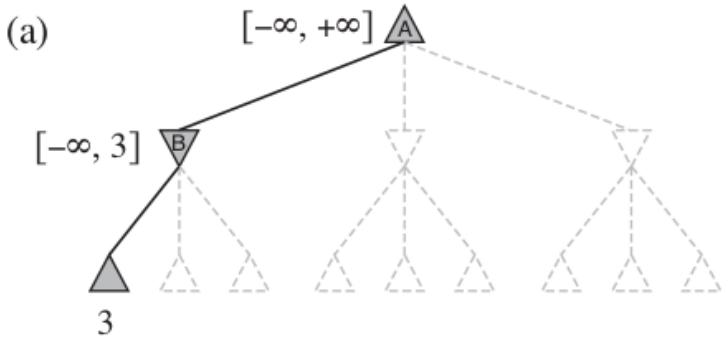
Źródło: CS221, Liang i Ermon

Mamy: $\max(3, \leq 2) = 3$

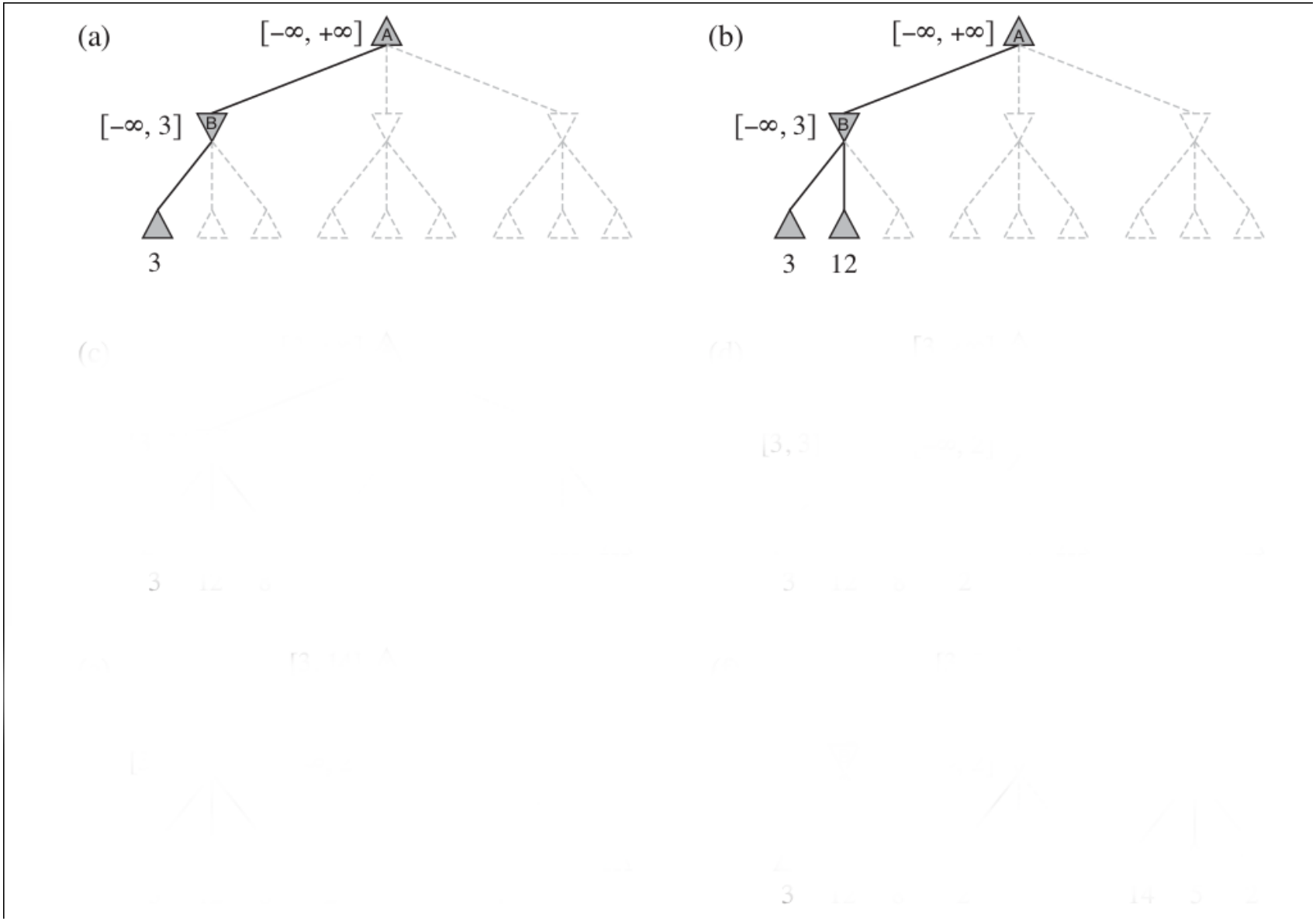
Obcinanie fragmentów drzew

- Jeżeli możemy udowodnić, że w jakimś poddrzewie nie ma optymalnej wartości, to możemy pominąć to poddrzewo.
- Będziemy pamiętać:
  - α – dolne ograniczenie dla węzłów MAX ($\geq \alpha$)
  - β – górne ograniczenie dla węzłow MIN ($\leq \beta$)

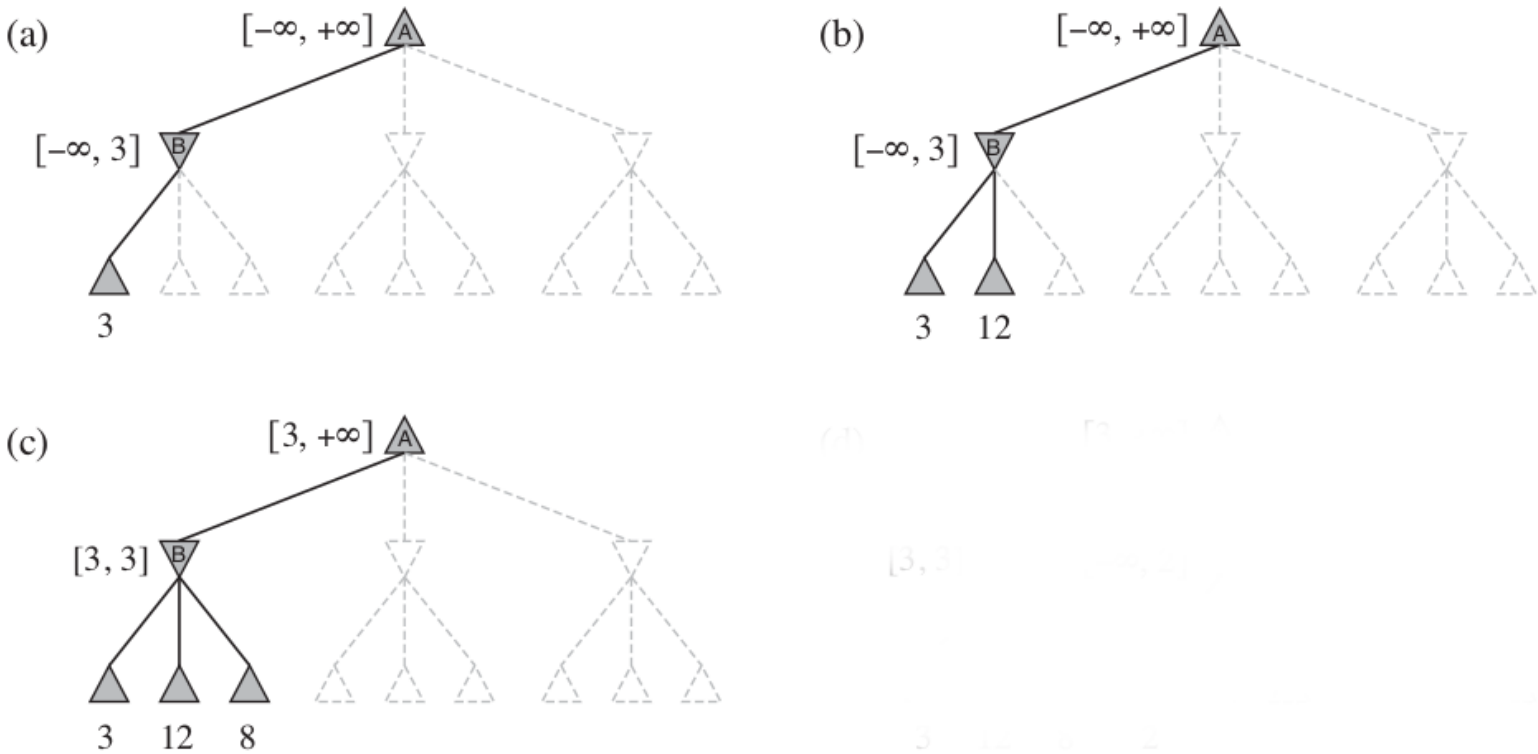
Alfa-Beta w akcji



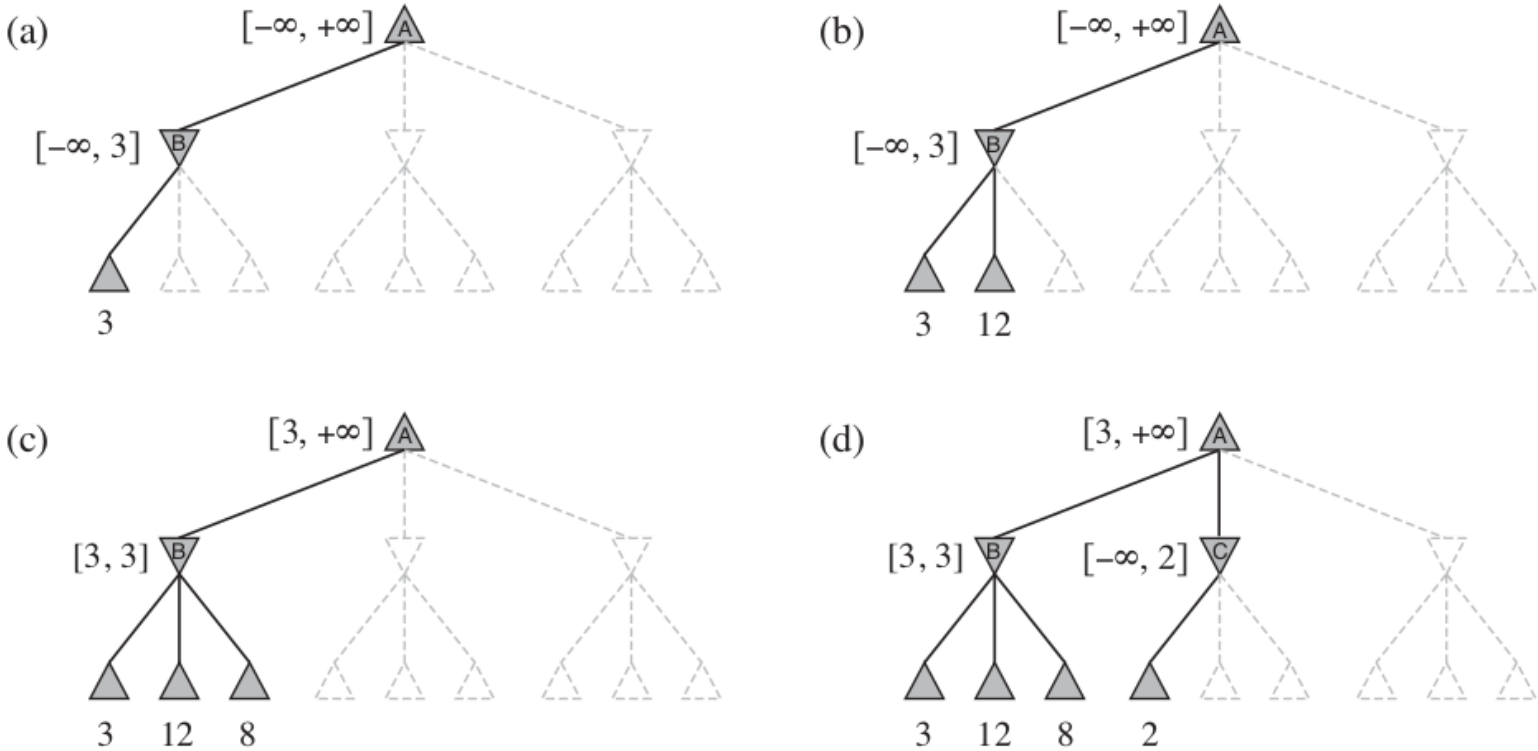
Alfa-Beta w akcji



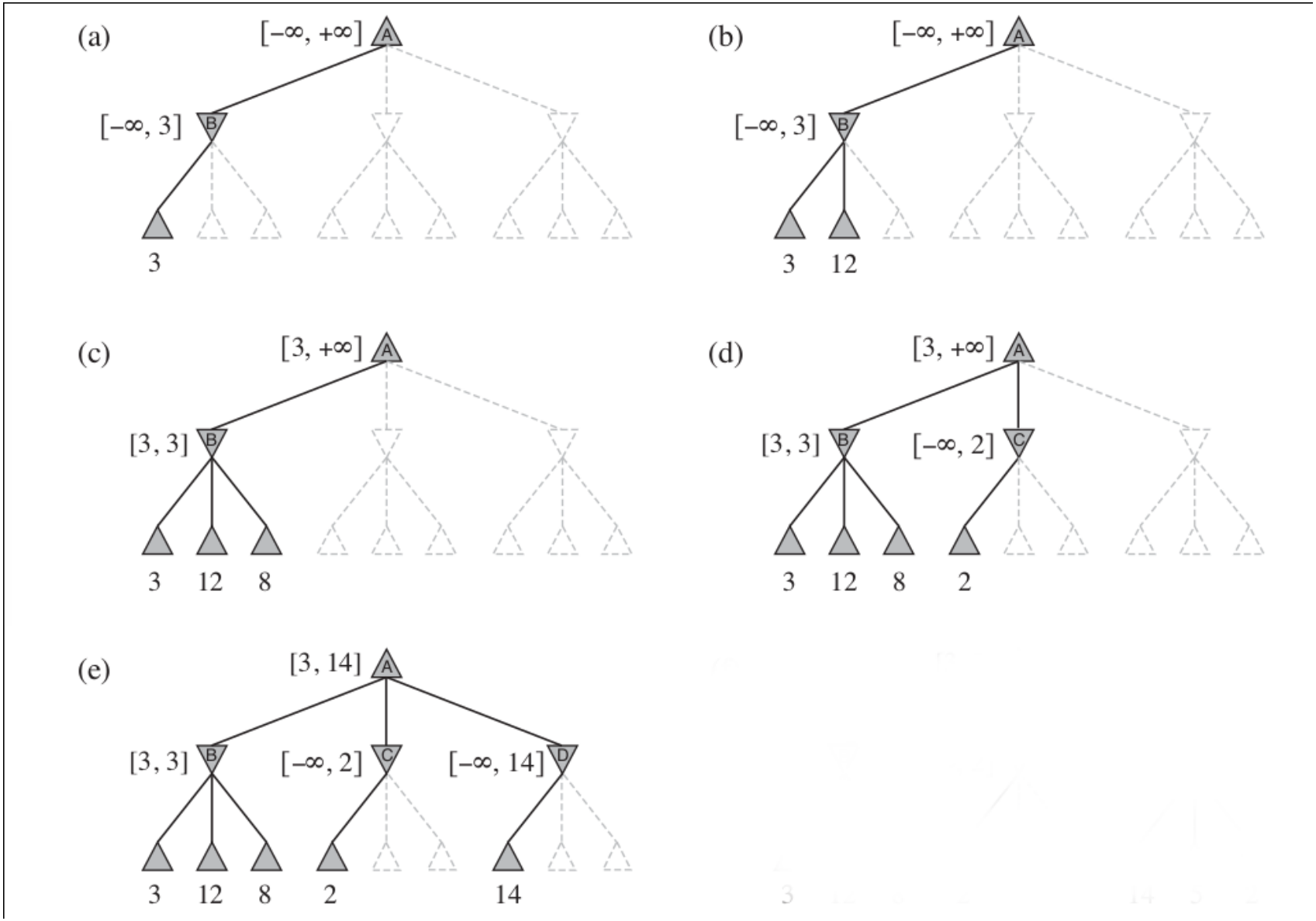
Alfa-Beta w akcji



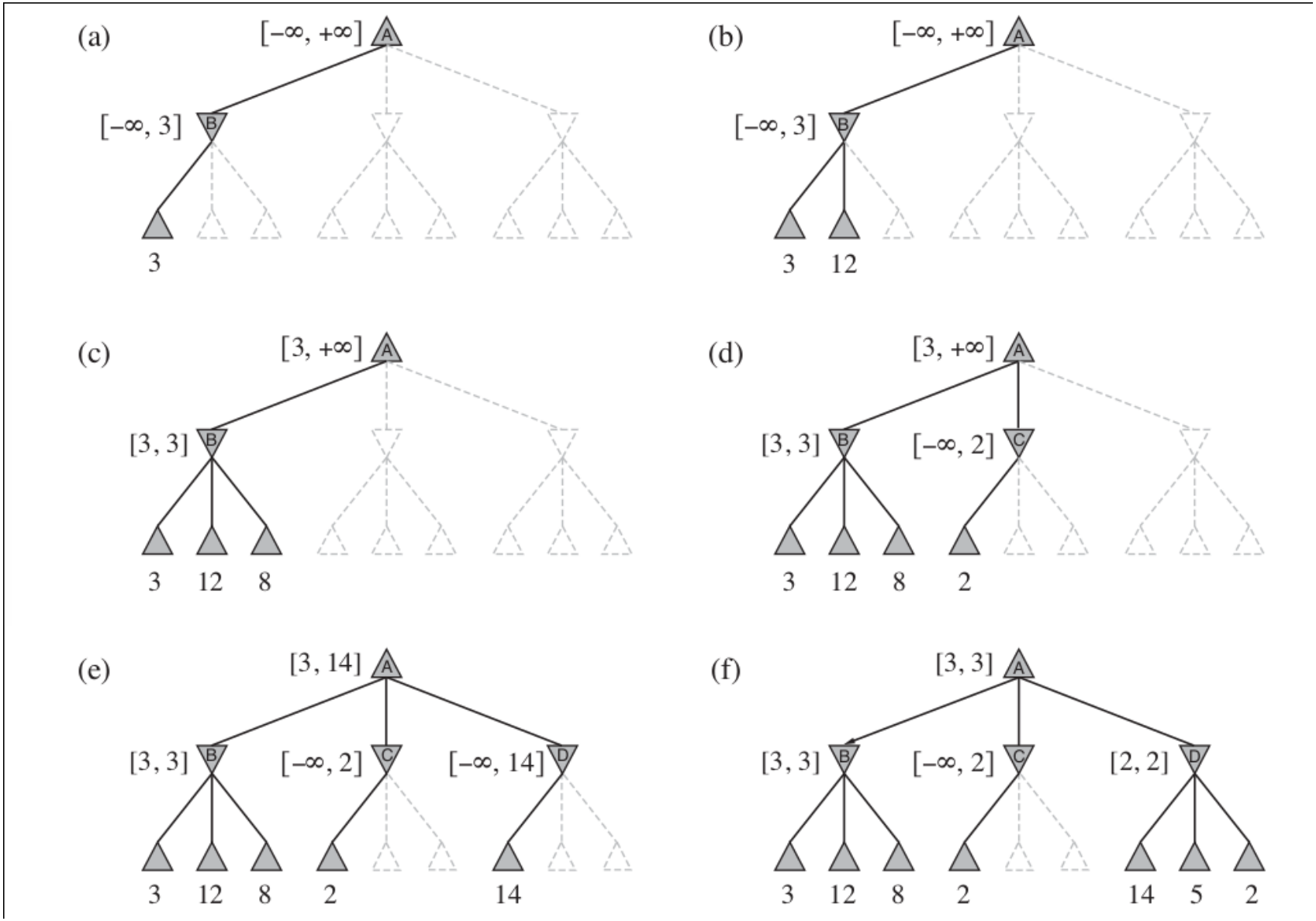
Alfa-Beta w akcji



Alfa-Beta w akcji



Alfa-Beta w akcji



Algorytm A-B

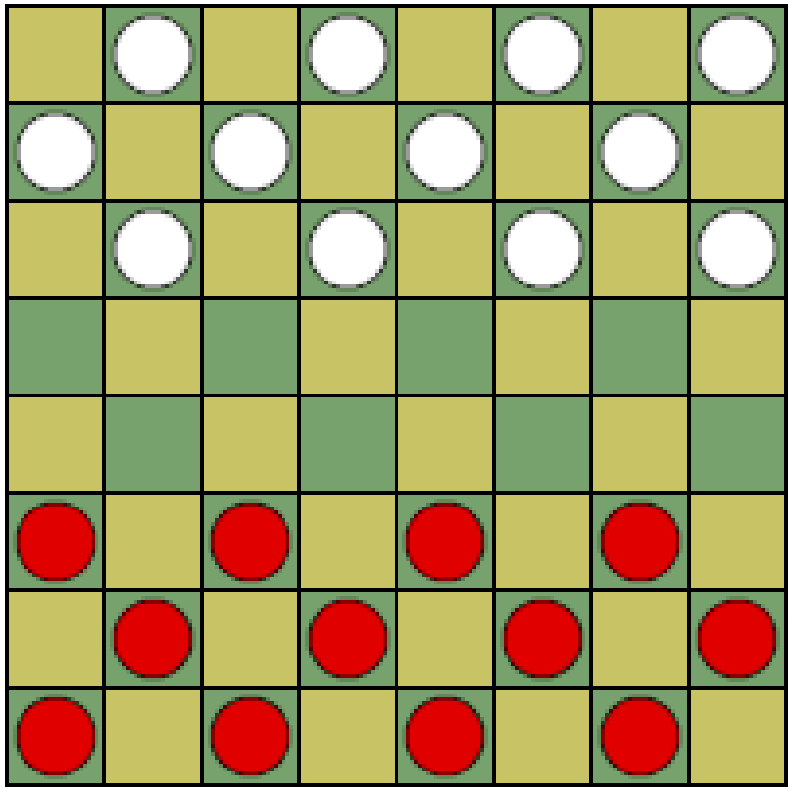
```
def max_value(state, alpha, beta):
    if terminal(state): return utility(state)
    value = -infinity

    for state1 in [result(a, state) for a in actions(state)]:
        value = max(value, min_value(state1, alpha, beta))
        if value >= beta:
            return value
        alpha = max(alpha, value)
    return value

def min_value(state, alpha, beta):
    if terminal(state): return utility(state)
    value = infinity

    for state1 in [result(a, state) for a in actions(state)]:
        value = min(value, max_value(state1, alpha, beta))
        if value <= alpha:
            return value
        beta = min(beta, value)

    return value
```

Kolejność węzłów

- Efektywność obcięć zależy od porządku węzłów.
- Dla losowej kolejności mamy czas działania $O(b^{2\times 0.75d})$ (czyli efektywne zmniejszenie głębokości do $\frac{3}{4}$)

Dobrym wyborem jest użycie funkcji `heuristic_value` do porządkowania węzłów.

Warcaby



- Ruch po skosie, normalne pionki tylko do przodu.
- Bicie obowiązkowe, można bić więcej niż 1 pionek. Wybieramy maksymalne bicie.
- Przemiana w tzw. damkę, która rusza się jak goniec.

Warcaby – uczenie się gry

- Pierwszy program, który „uczył” się gry, rozgrywając partie samemu ze sobą.
- Autor: Arthur Samuel, 1965

Przyjrzyjmy się ideom wprowadzonym przez Samuela.

Program Samuela

1. Alpha-beta search (po raz pierwszy!) i spamiętywanie pozycji
2. Unikanie zwycięstwa i przyśpieszanie porażki: mając do wyboru dwa ruchy o tej samej ocenie:
   - wybieramy ten z dłuższą grą (jeżeli przegrywamy)
   - a ten z krótszą (jeżeli wygrywamy)

Idea uczenia przez granie samemu ze sobą

Wariant 1

Patrzymy na pojedynczą sytuację i próbujemy z niej coś wydedukować.

Wariant 2

Patrzymy na pełną rozgrywkę i:

a) Jeżeli wygraliśmy, to znaczy, że nasze ruchy były dobre a przeciwnika złe

b) W przeciwnym przypadku – odwrotnie.

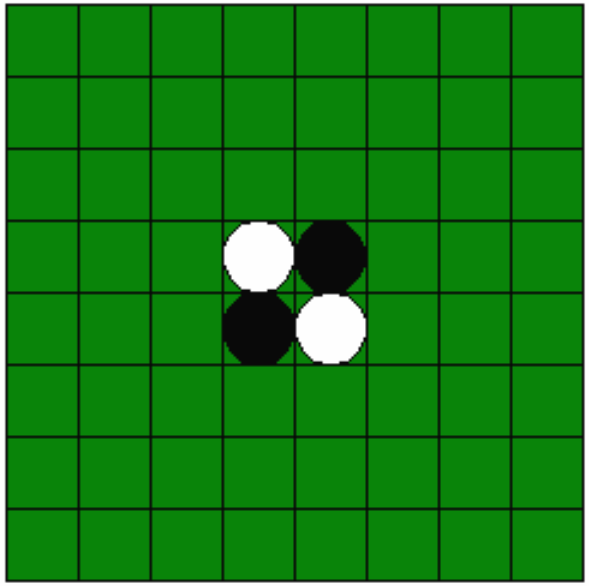
W programie Samuela użyty był wariant pierwszy. Program starał się tak modyfikować parametry funkcji uczącej, żeby możliwie przypominała **minimax** dla głębokości 3 z bardzo prostą funkcją oceniającą (liczącą bierki).

Reversi

- Gra znana od końca XIX wieku.
- Od około 1970 roku pod nazwą Othello.

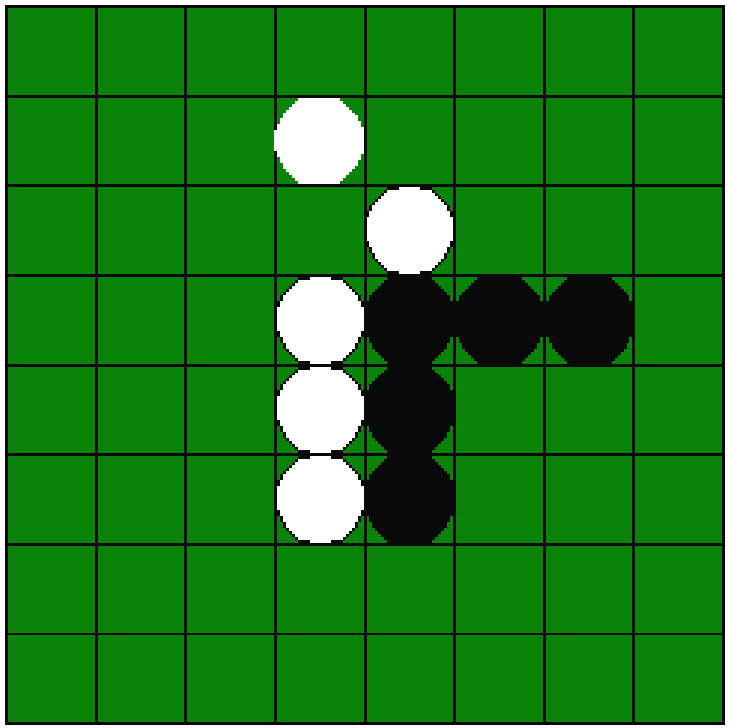
Nadaje się dość dobrze do prezentacji pewnych idei związanych z grami: uczenia i Monte Carlo Tree Search.

Reversi. Zasady



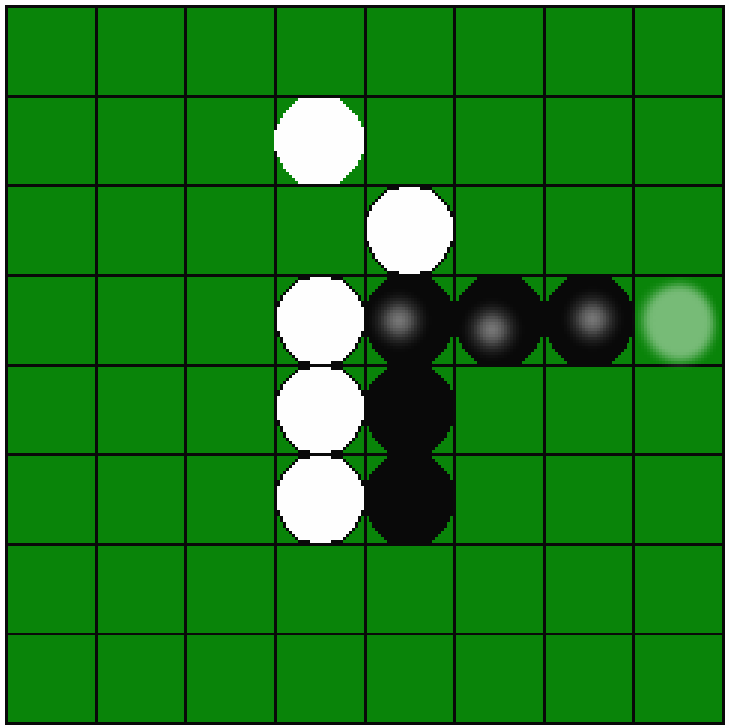
- Zaczynamy od powyższej pozycji.
- Gracze na zmianę dokładają pionki.
- Każdy ruch musi byś biciem, czyli okrążeniem pionów przeciwnkika w wierszu, kolumnie lub linii diagonalnej.
- Zbite pionki zmieniają kolor (możliwe jest bicie na więcej niż 1 linii).
- Wygrywa ten, kto pod koniec ma więcej pionków.

Reversi. Ruchy



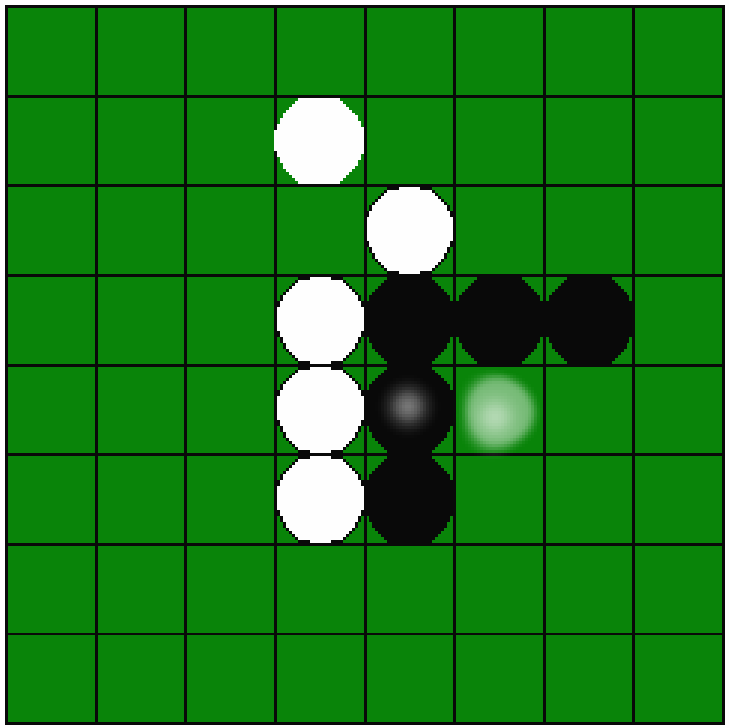
Ruch przypada na białego.

Reversi. Ruchy



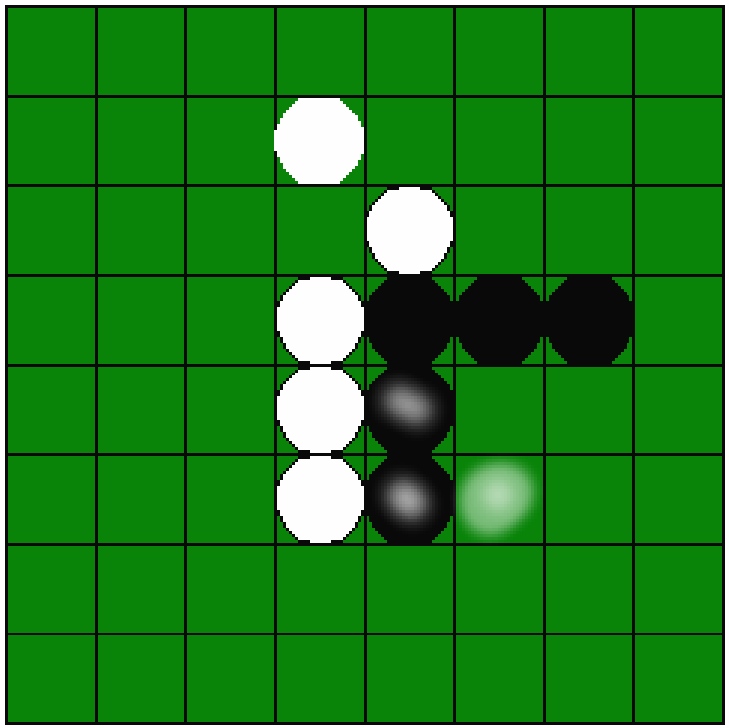
Bicie w poziomie

Reversi. Ruchy



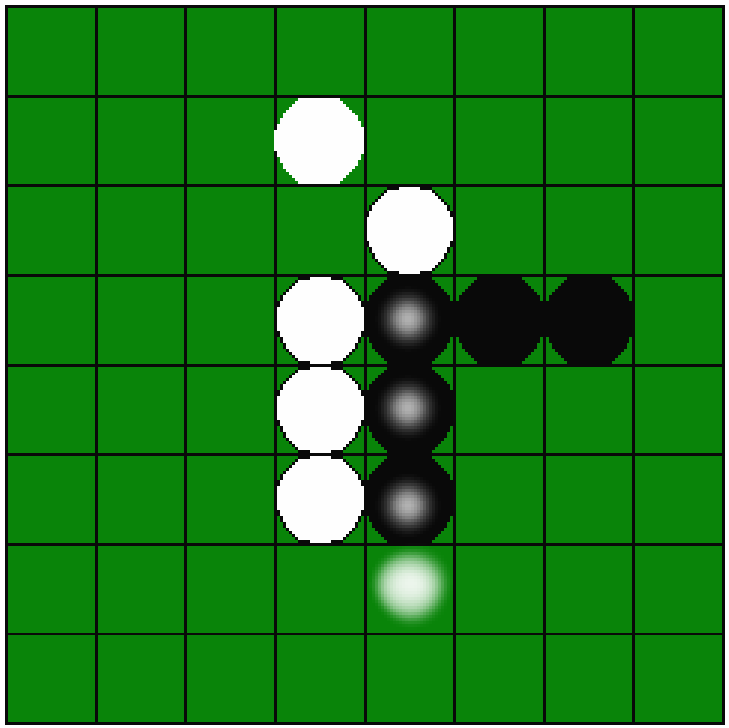
Bicie w poziomie

Reversi. Ruchy



Bicie w poziomie i po skosie

Reversi. Ruchy



Bicie w pionie

Przykładowa gra

- Popatrzmy szybko na przykładową grę.
- Biały: minimax, głębokość 3, funkcja oceniająca = balans pionków
- Czarny: losowe ruchy

Prezentacja: reversi_show.py

Przykładowa gra. Wnioski

Wniosek 1

Gracz losowy działa całkiem przyzwoicie. Może to świadczyć o sensowności oceny sytuacji za pomocą symulacji.

Wniosek 2

Jest wyraźna potrzeba nauczenia się sensowniejszej funkcji oceniającej.

Prosty eksperyment

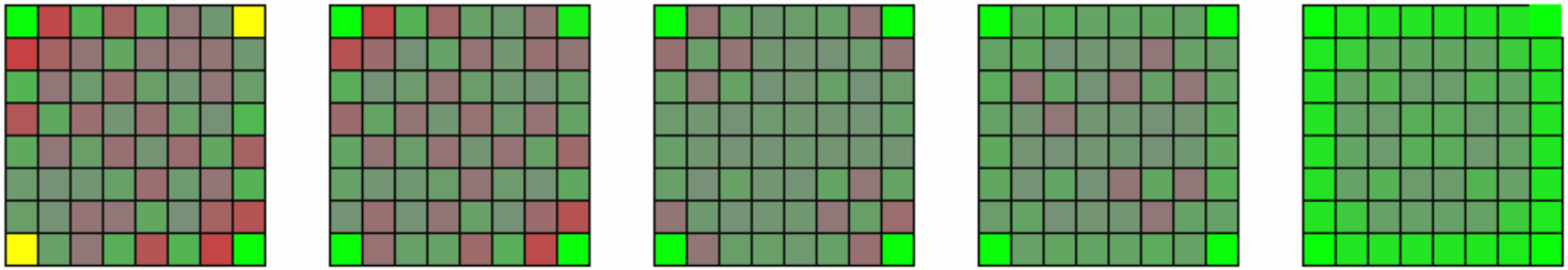
Cel

Ocena wartości pól w różnych momentach gry (pod koniec wiadomo jaka).

1. Wykonujemy losowych K-ruchów. Będziemy oceniać wartość pól po K ruchach.
2. Rozgrywamy partię po tych K ruchach:
   a. Białe wygrały: zwiększamy trochę wartość pól zajętych przez białe, zmniejszamy wartość pól zajętych przez czarne.
   b. Czarne wygrały: postępujemy odwrotnie.

Wyniki eksperymentu

- Zielone – pozytywne pola, warto na nich mieć pionka w momencie K.
- Czerwone – tych pól powinniśmy raczej unikać (w momencie K), mają bowiem wartość ujemną, czyli utrzymując je zajęte, częściej przegrywamy niż wygrywamy.



Wyniki dla $K = 6, 10, 30, 40, 56$

Uczenie się

- Spróbujemy usystematyzować nasze intuicje związane z uczeniem.
- Co wiemy:
  a. Mamy przykłady, próbujemy je uogólnić.
  b. Jedno z podstawowych zadań: klasyfikacja, czyli przypisanie przypadkowi jego klasy.
  c. Przykłady:
     - Ocena, czy mail należy do spam czy też nie-spam.
     - Wybór rasy dla zdjęcia psa
     - Czy napis jest adresem e-mail, url-em, nazwą firmy, imieniem i nazwiskiem, czymś innym?

Regresja

- Oczekiwanym wynikiem może być liczba rzeczywista.
- Przykłady:
  - predycja ceny nieruchomości,
  - ocena masy ciała (gdy znamy płeć i wzrost),
  - przewidywanie zużycia wody (dla MPWiK), gdy znamy temperaturę i dzień tygodnia

Cechy (wektor cech)

- Abstrakcyjny obiekt możemy zamienić na wektor cech.
- Dla (zabawkowego) klasyfikatora **czy-email?**, możemy mieć:
  - Czy długość większa od 10?
  - Jaki procent znaków to znaki alfanumeryczne?
  - Czy zawiera @?
  - Czy kończy się na .com (i tak dalej)

Cechami mogą być też na przykład wartości składowych pikseli, kolejne wartości pliku wave, zbiory pomiarów wszystkich wodomierzy z ostatniej doby, itd.

Dla obiektu x wektor cech oznaczamy często jako $(\phi_1(x), \ldots, \phi_n(x))$.

Dane i uczenie

- Dane uczące – zbiór przykładów, często w postaci:

  (wektor-cech1, wynik1)
  (wektor-cech2, wynik2)
  (wektor-cech3, wynik3)
  ...

Uczenie funkcji liniowej

- Klasyfikacji (czy regresji) dokonujemy na bazie wartości:

$$\sum_{i=0}^{N} w_i \phi_i(x)$$

- Możemy zdefiniować zadanie uczenia (regresji) dla Reversi:

  Widząc sytuację na planszy w ruchu 20 postaraj się przewidzieć zakończenie gry.

- Cechy: binarne cechy mówiące o zajętości pola.

Funkcja straty

Definicja

Funkcja straty (loss) opisuje, jak bardzo niezadowoleni jesteśmy z działania naszego mechanizmu (klasyfikatora, przewidywacza wartości).

Funkcja straty jest określona na: danych uczących (x,y) oraz wagach (parametrach klasyfikatora).

Przykładowa funkcja straty: błąd średniokwadratowy

Funkcja straty (2)

Średniokwadratowa funkcja straty

$$\text{TrainLoss}(\mathbf{w}) = \frac{1}{|D_{\text{train}}|} \sum_{(x,y) \in D_{\text{train}}} (f_{\mathbf{w}}(x) - y)^2$$

Wariant liniowy:

$$f_{\mathbf{w}}(x) = \sum_{i=0}^{N} w_i \phi_i(x)$$

Gradient

- Gradient jest wektorem pochodnych cząstkowych.
- Wskazuje kierunek największego wzrostu funkcji.
- Policzmy gradient na tablicy, dla powyżej zdefiniowanej funkcji kosztu.