Gry: efektywność i symulacje

Paweł Rychlikowski

Instytut Informatyki UWr

10 kwietnia 2024

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).
- Czasem do stanu możemy dojść na wiele sposobów (zwłaszcza, jak ruchy są od siebie niezależne)
- 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 jest na g6 (WB-G6), czarny król jest 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:
    nowy-kod = stary-kod **xor** wk-a4 **xor** wk-b5
    to ruch białego króla z a4 na b5

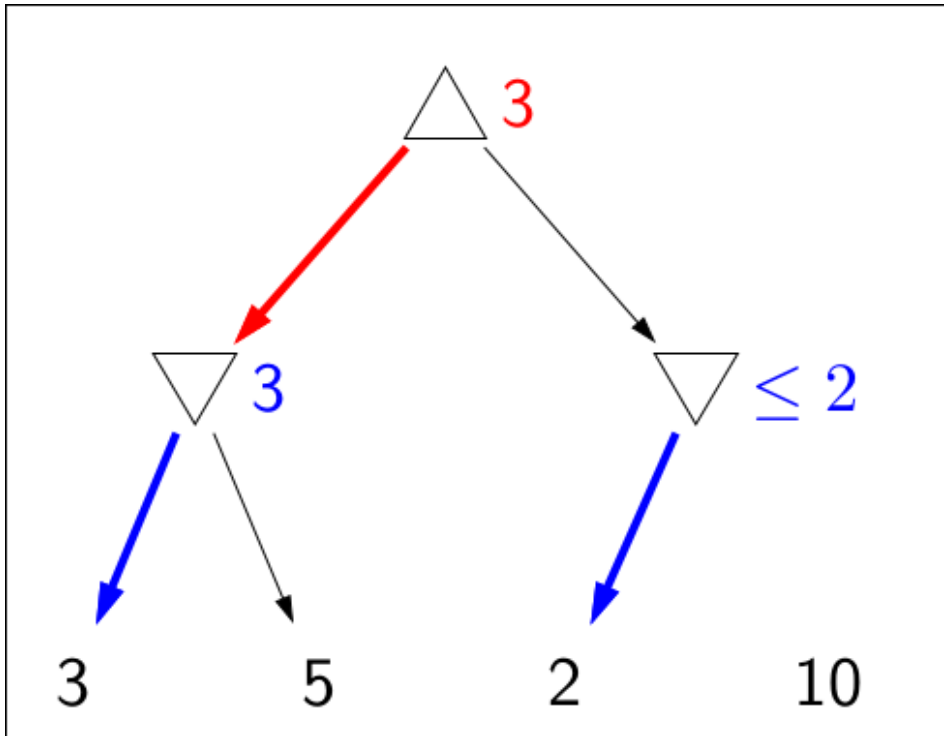
Uwaga

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

Obcinanie fragmentów drzew

Idea

nie zawsze musimy przeglądać całe drzewo, żeby wybrać optymalną ścieżkę



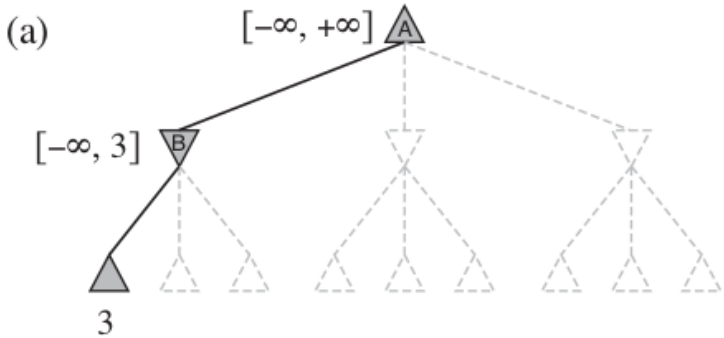
Źródło: CS221, Liang i Ermon

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

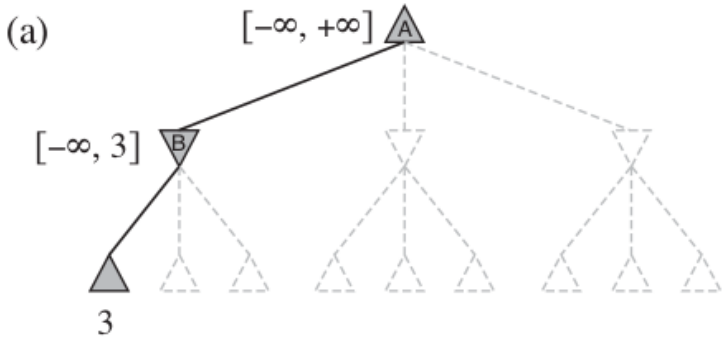
Obcinanie fragmentów drzew

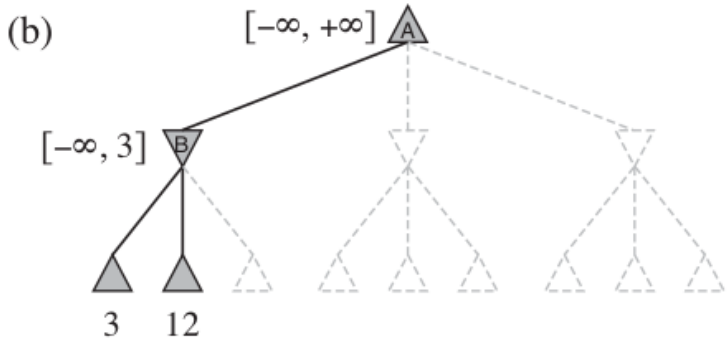
- Jeżeli możemy udowodnić, że w jakimś poddrzewie nie ma optymalnej wartości, to możemy pominąć to poddrzewo.
- Będziemy pamiętać:
  - α – dolne ograniczenie dla węzłów MAX ($\geq \alpha$)
  - β – górne ograniczenie dla węzłow MIN ($\leq \beta$)

Alfa-Beta w akcji

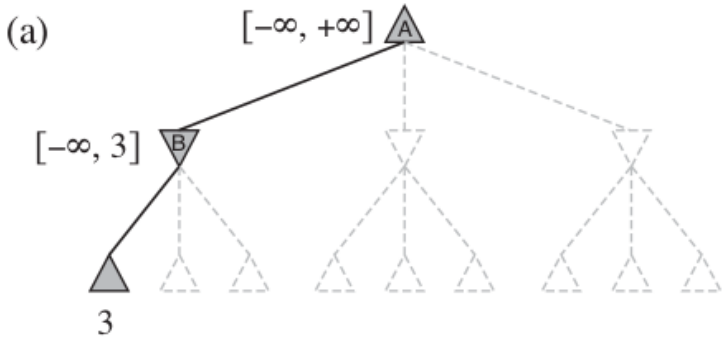


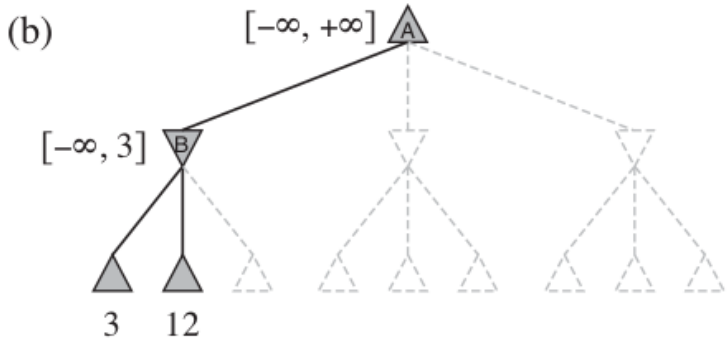
Alfa-Beta w akcji

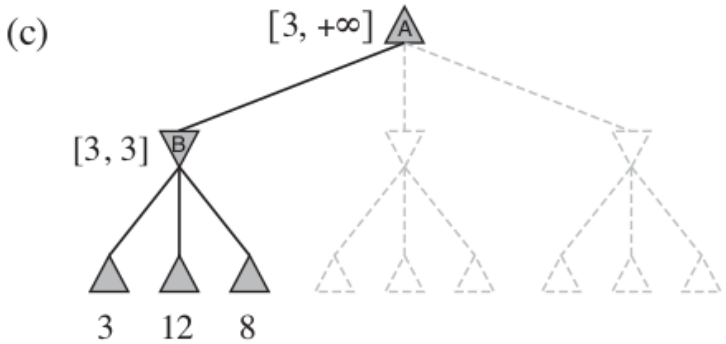




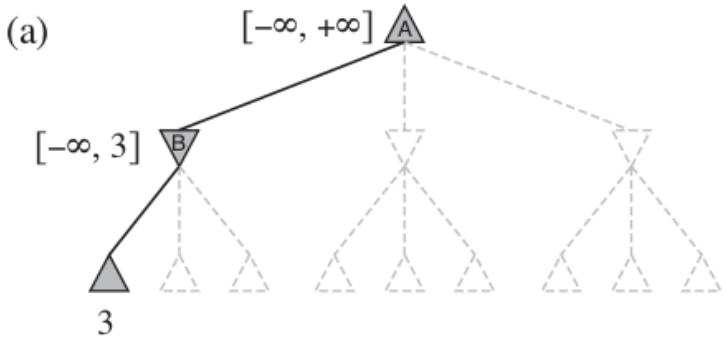
Alfa-Beta w akcji

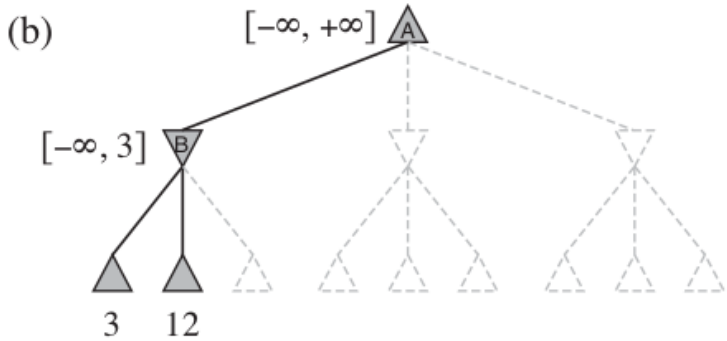


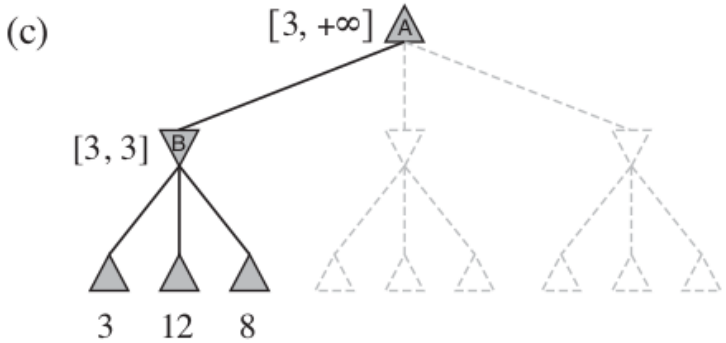


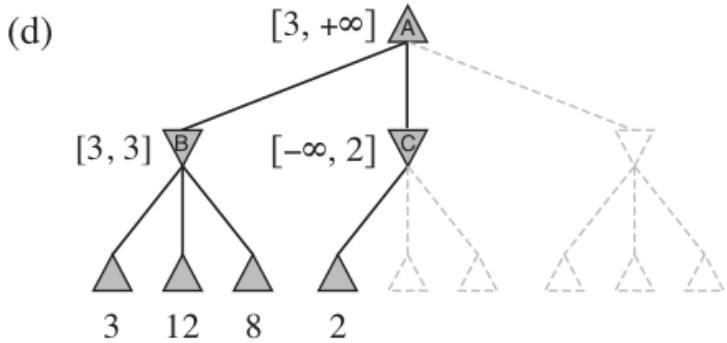


Alfa-Beta w akcji

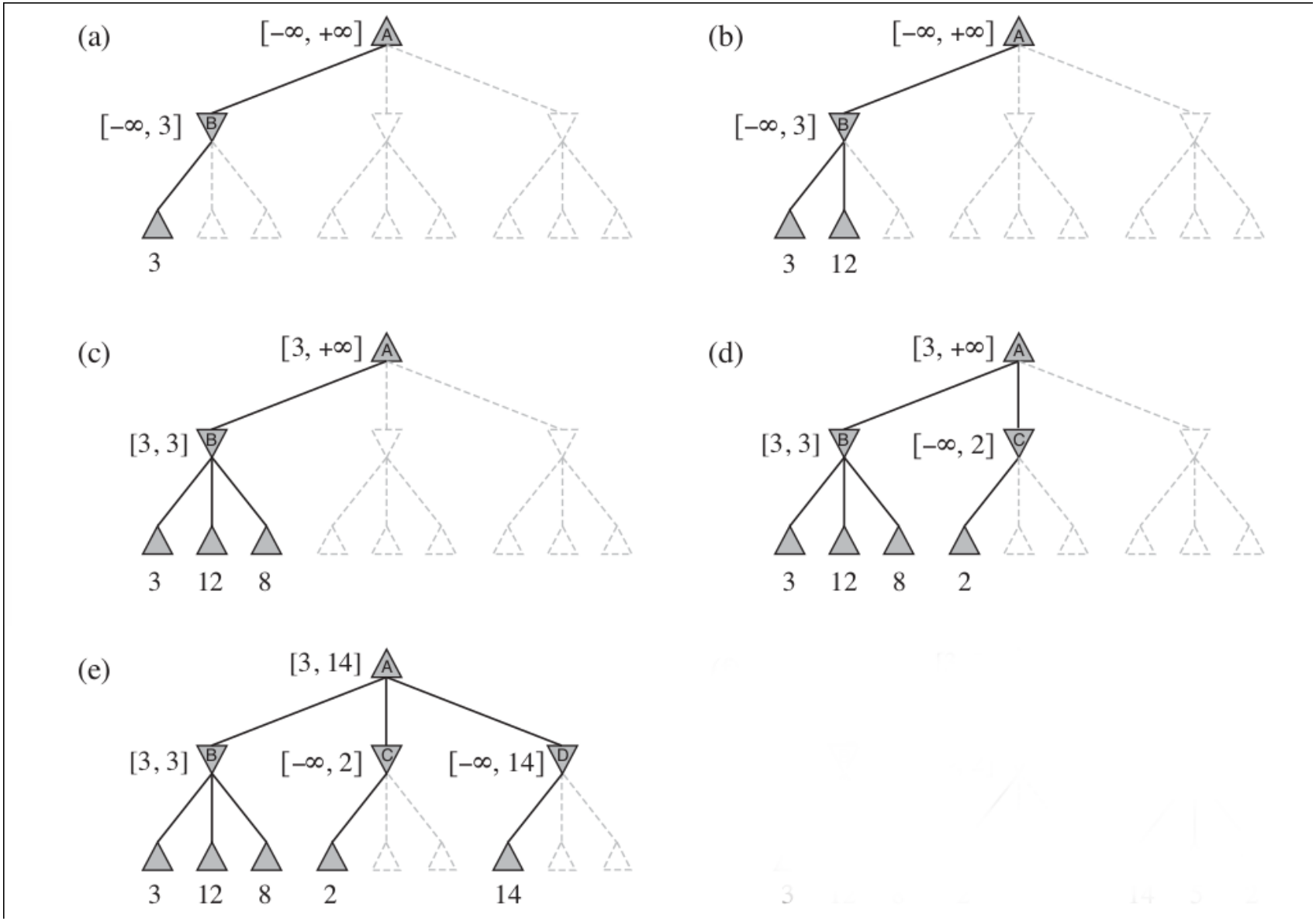




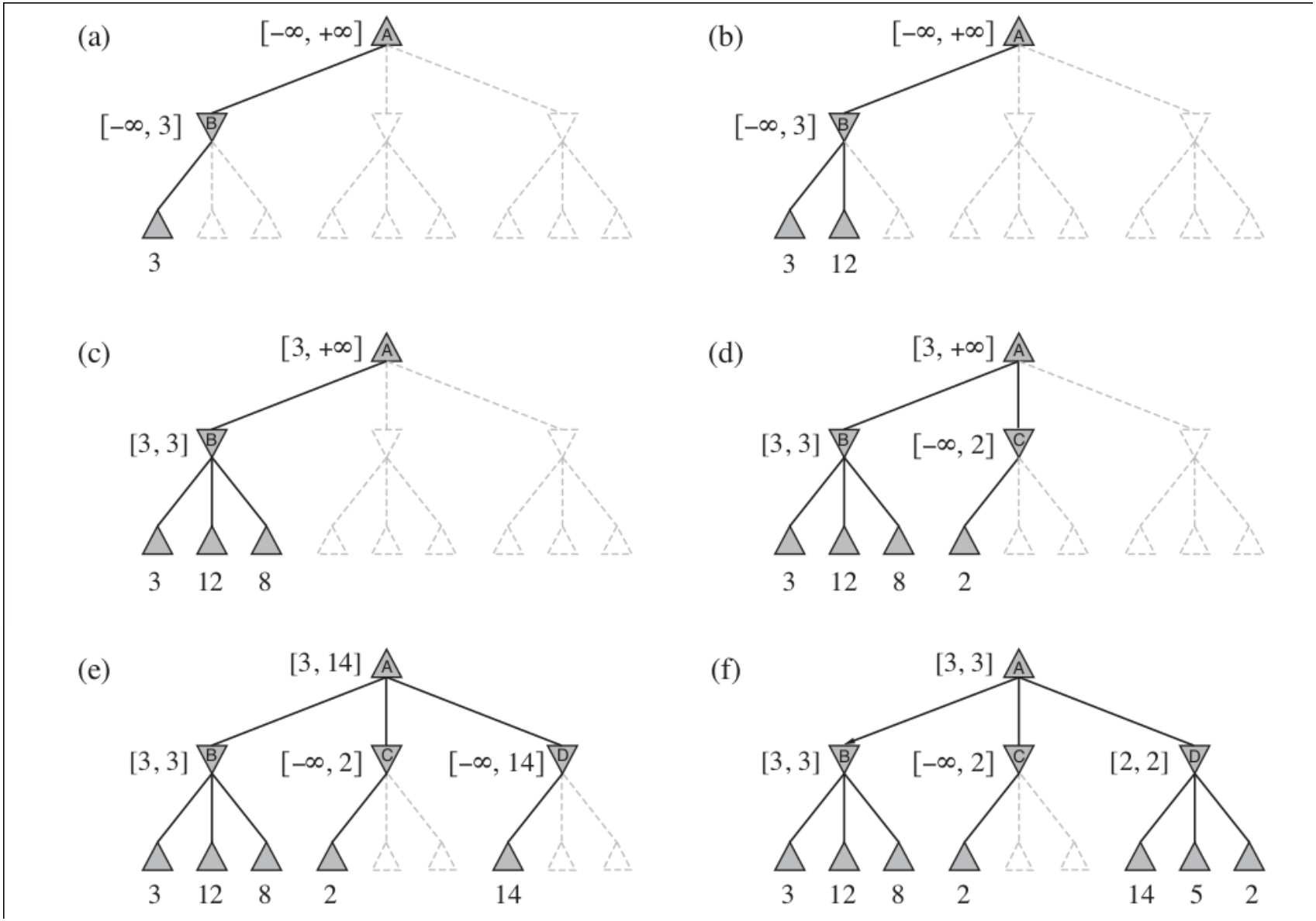




Alfa-Beta w akcji



Alfa-Beta w akcji



Obcinanie fragmentów drzew. Przypomnienie slajdu

- Będziemy pamiętać:
  - α – dolne ograniczenie dla węzłów MAX ($\geq \alpha$)
  - β – górne ograniczenie dla węzłow MIN ($\leq \beta$)

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

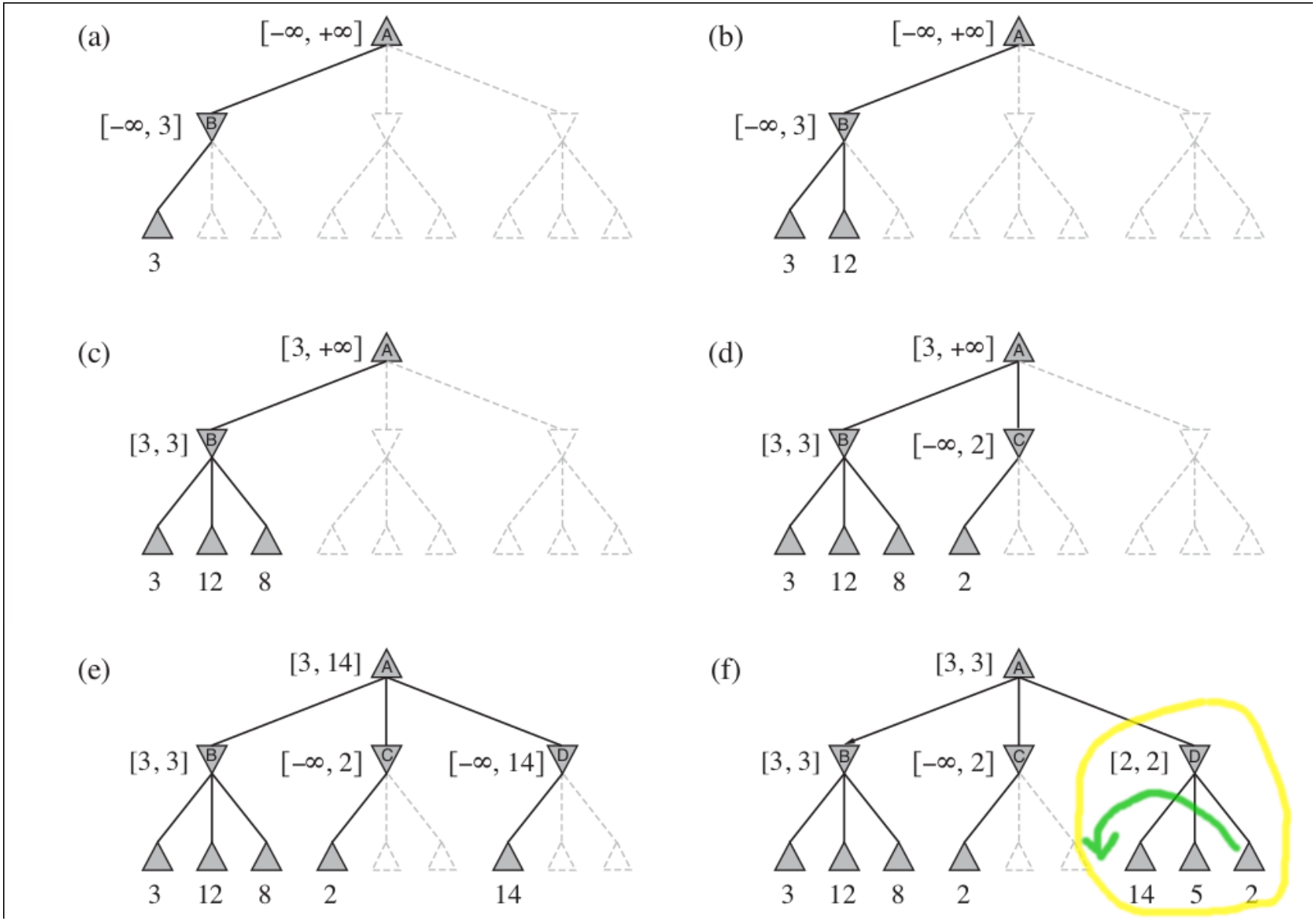
- Efektywność obcięć zależy od porządku węzłów.
- Dla losowej kolejności mamy czas działania $O(b^{2\times 0.75d})$ (czyli efektywne zmniejszenie głębokości do $\frac{3}{4}$)

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

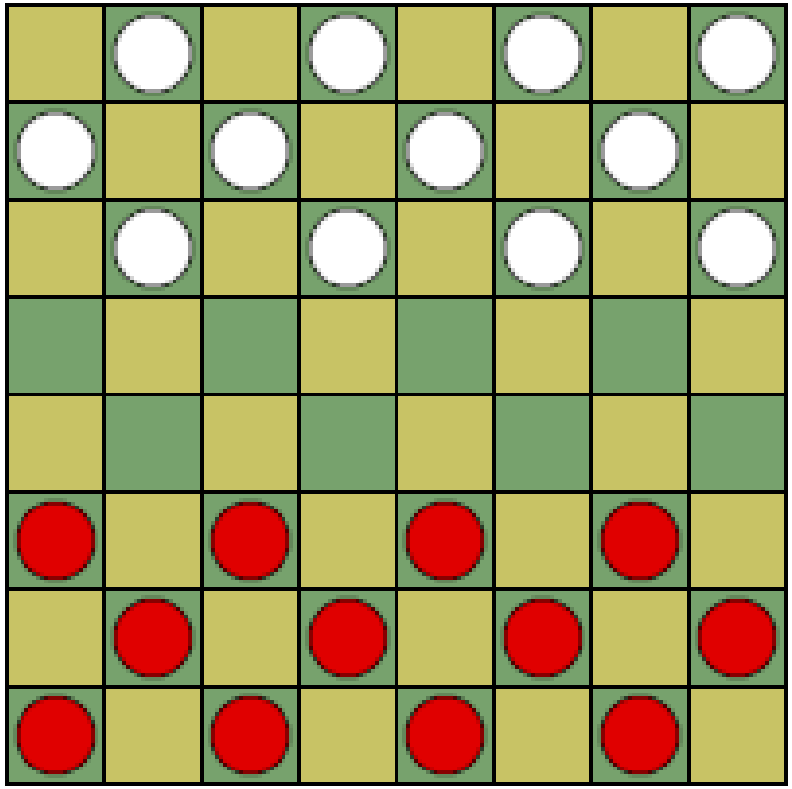
Uwaga

Warto porządkować węzły jedynie na wyższych piętrach drzewa gry!

Zmiana kolejności wpływa na efektywność



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. Przyśpieszanie zwycięstwa i oddalanie 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.

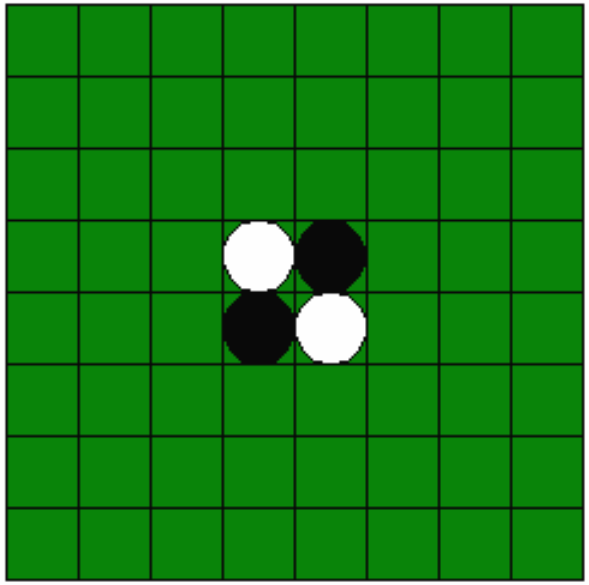
W programie Samuela użyty był wariant pierwszy. Program starał się tak modyfikować parametry funkcji uczącej, żeby możliwie przypominała **minimax** dla głębokości 3 z bardzo prostą funkcją oceniającą (liczącą bierki).

Reversi

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

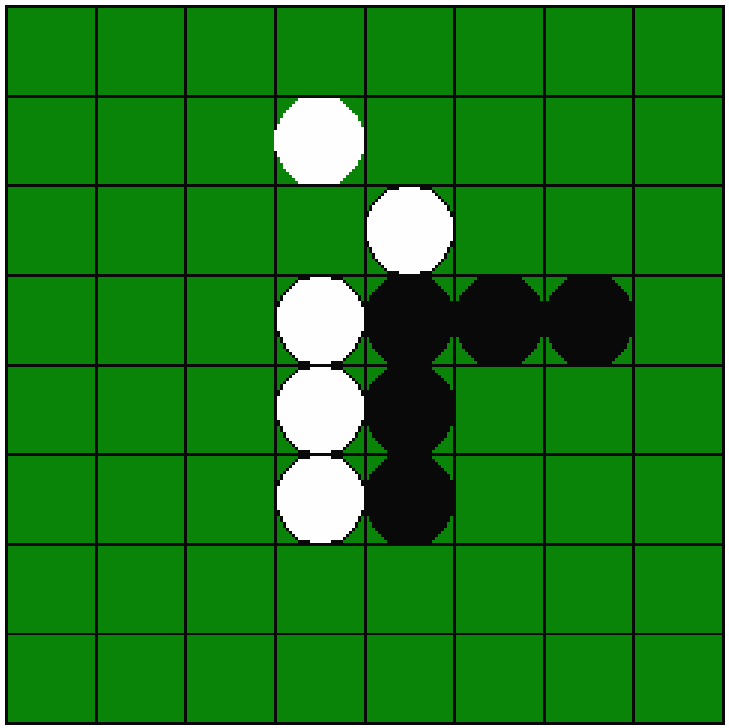
Nadaje się dość dobrze do prezentacji pewnych idei związanych z grami: uczenia i Monte Carlo Tree Search.

Reversi. Zasady



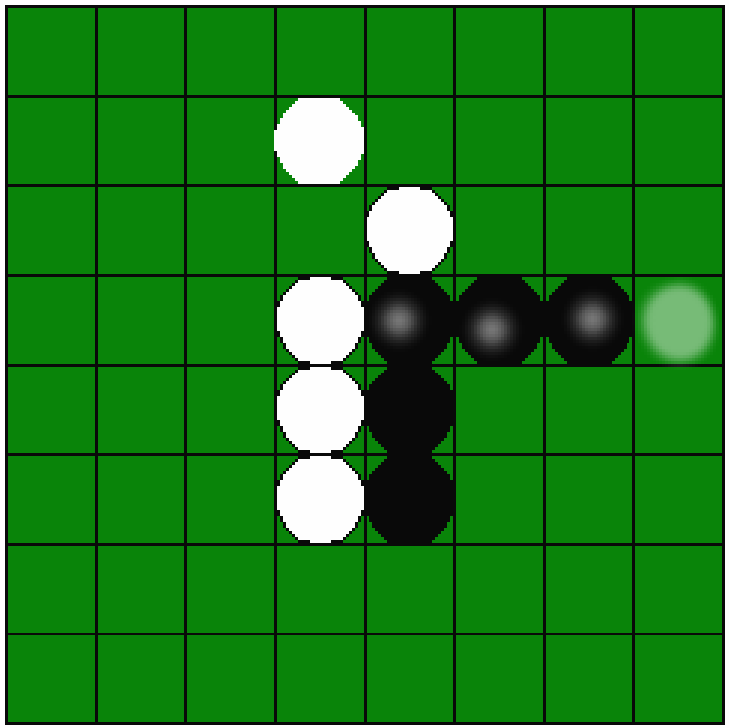
- Zaczynamy od powyższej pozycji.
- Gracze na zmianę dokładają pionki.
- Każdy ruch musi byś biciem, czyli okrążeniem pionów przeciwnkika w wierszu, kolumnie lub linii diagonalnej.
- Zbite pionki zmieniają kolor (możliwe jest bicie na więcej niż 1 linii).
- Wygrywa ten, kto pod koniec ma więcej pionków.

Reversi. Ruchy



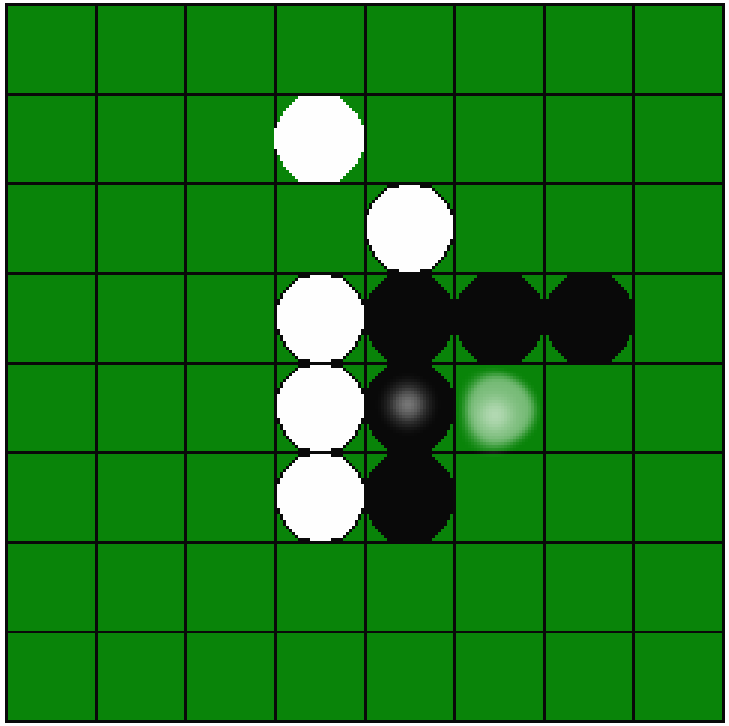
Ruch przypada na białego.

Reversi. Ruchy



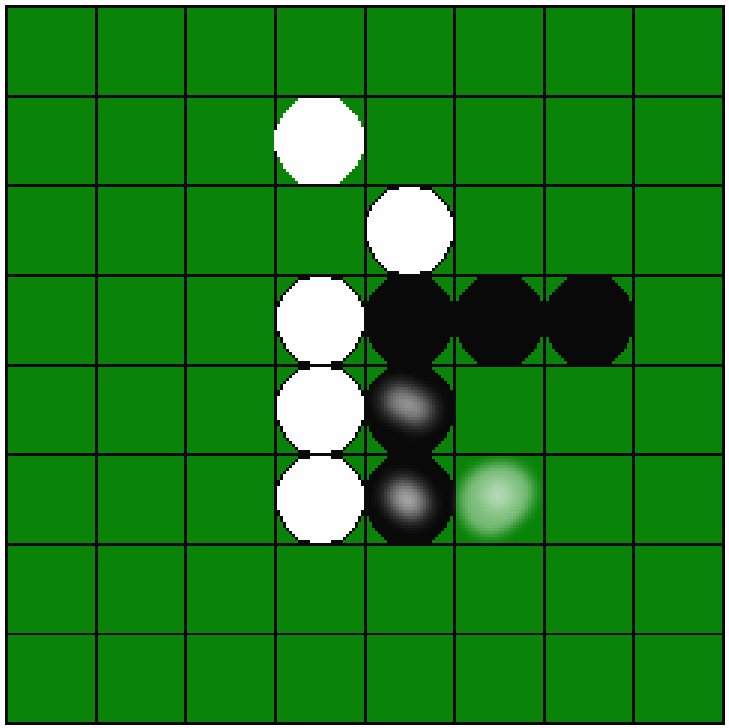
Bicie w poziomie

Reversi. Ruchy



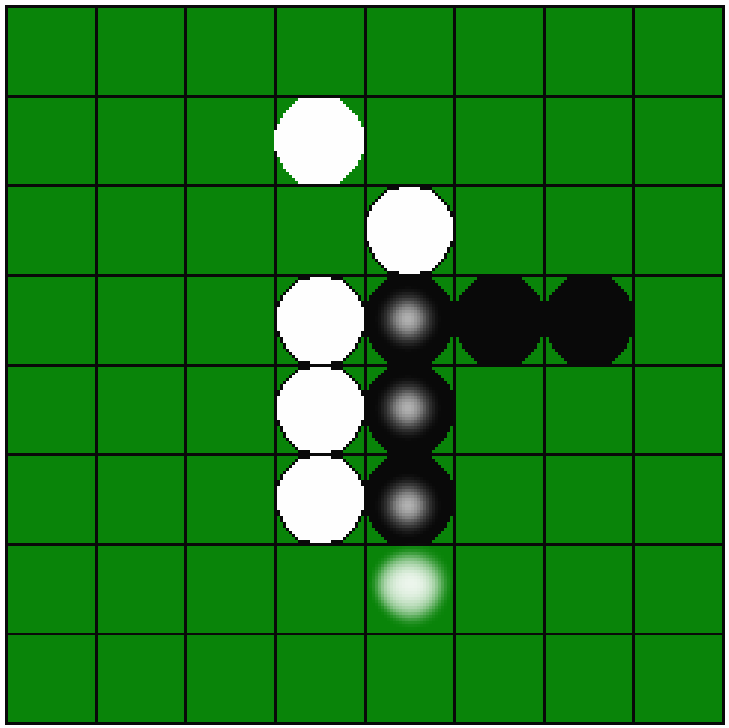
Bicie w poziomie

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_original.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.

Eksploracja i eksploatacja

Wariant *życiowy*

Jesteśmy na wakacjach, jemy obiad w restauracji. Nawet smakowało. Powtarzamy, czy szukamy innego miejsca?

- Standardowy dylemat agenta działającego w nieznanym środowisku:
  1. Maksymalizować swoją korzyść biorąc pod uwagę aktualną wiedzę o świecie.
  2. Starać się dowiedzieć więcej o świecie, być może ryzykując nieoptymalne ruchy.
- Pierwsza strategie to eksploatacja, druga to eksploracja.

Jednoręki bandyta

Źródło: Wikipedia

Po pociągnięciu za rączkę, pojawia się wzorek, który (potencjalnie) oznacza naszą niezerową wypłatę.

Wieloręki bandyta

- Mamy wiele tego typu maszyn.
- Możemy zapomnieć o wzorkach, maszyny po prostu generują wypłatę, zgodnie z nieznanym rozkładem.
- Znajomy właściciel kasyna wpuścił nas na kwadrans do sali z takimi automatami. Jak gramy?
- Bardzo wyraźnie widać dylemat eksploracja vs eksploatacja.

Wieloręki bandyta. Przykładowe strategie

- **Zachłanna**: każda rączka po razie, a następnie... ta która dała najlepszy wynik.
  - **Lepiej:** najlepszy średni wynik do tej pory
- ε**-zachłanna**: rzucamy monetą. Z $p = \varepsilon$ wykonujemy ruch losową rączką, z $p = 1 - \varepsilon$ – wykonujemy ruch rączką, która ma najlepszy średni wynik do tej pory.
- **Optymistyczna wartość początkowa**: inny sposób na zapewnienie eksploracji. Na początku każdy wybór obniża atrakcyjność danego bandyty.

Upper Confidence Bound

- Wybieramy akcję a (bandytę) maksymalizującą:

$$Q_t(a) + c\sqrt{\frac{\ln t}{N_t(a)}}$$

gdzie: Q_t to uśredniona wartość akcji do momentu t, N_t – ile razy dana akcje była wybierana (do momentu t)

- Zwróćmy uwagę, że jak akcja nie jest wybierana, to prawy składnik powoli rośnie. Akcja wybierana natomiast traci „premię eksploracyjną", na początku w szybkim tempie (wzrost mianownika).

Uwaga

Bardzo powszechnie używana strategia! (np. w AlphaGo)

Monte Carlo Tree Search

Algorytm odpowiedzialny za przełom w:

a. W grze w Go
b. W General Game Playing

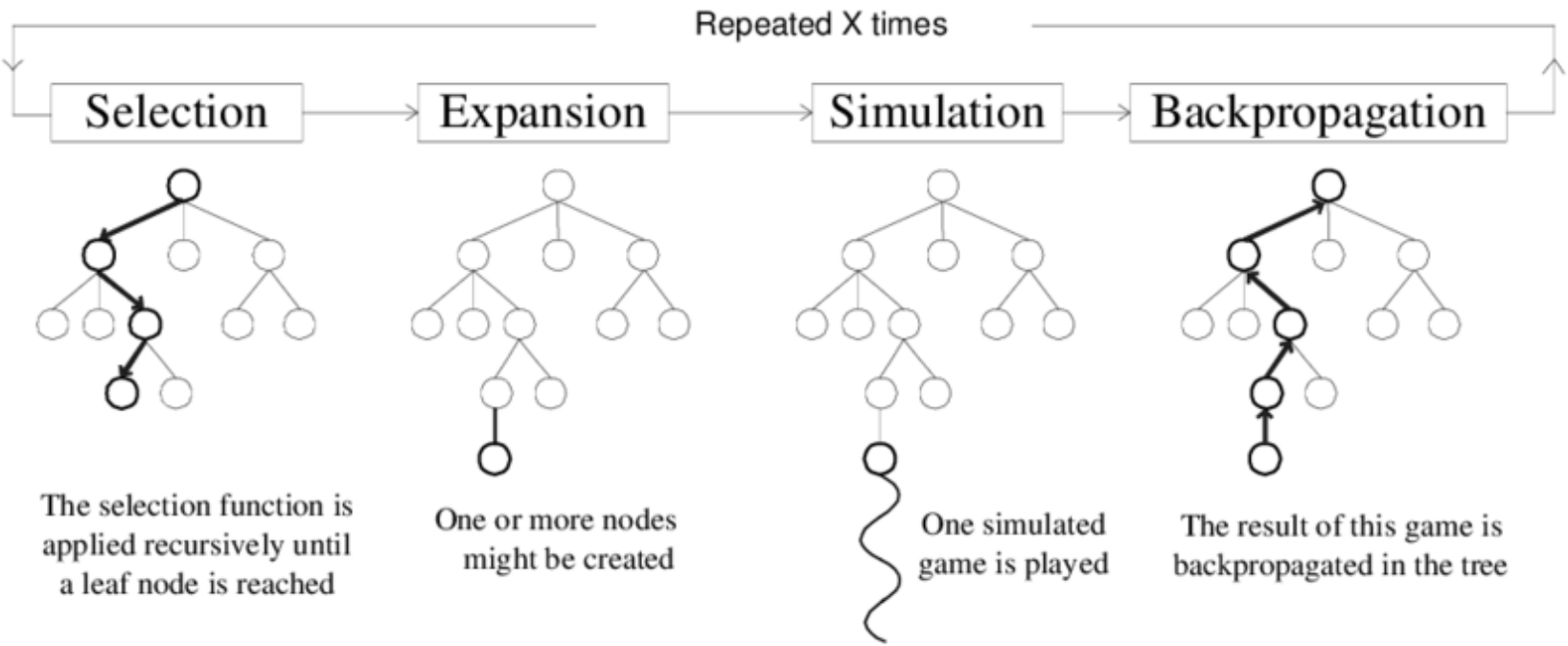
Główne idee

1. Oceniamy sytuację wykonując symulowane rozgrywki.
2. Budujemy drzewo gry (na początku składające się z jednego węzła – stanu przed ruchem komputera)
3. Dla każdego rozwiniętego węzła utrzymujemy statystyki, mówiące o tym, kto częściej wygrywał gry rozpoczynające się w tym węźle
4. Selekcję wykonujemy na każdym poziomie (UCB), na końcu rozwijamy wybrany węzeł dodając jego dzieci i przeprowadzając rozgrywkę.

MCTS. Podstawowe operacje

1. **Selection**: wybór węzła do rozwinięcia
2. **Expansion**: rozwinięcie węzła (dodanie kolejnych stanów)
3. **Simulation**: symulowana rozgrywka (zgodnie z jakąś polityką), zaczynające się od wybranego węzła
4. **Backup**: uaktualnienie statystyk dla rozwiniętego węzła i jego przodków

MCTS. Rysunek



Inna opcja

Rozwinięcie to dodanie wszystkich dzieci i (ewentualnie) przeprowadzenie dla nich po jednej symulowanej rozgrywce (powyższy rysunek zakłada rozwinięcie częściowe, wówczas dochodząc do węzła kolejny raz powinniśmy wziąć kolejny ruch, aż do uzyskania rozwinięcia pełnego).

MCTS. Dodatkowe uwagi

- Rozgrywka nie musi być prostym losowaniem, p-stwo ruchu może zależeć od jego (szybkiej!) oceny.
- Im więcej symulacji, tym lepsza gra – precyzyjne sterowanie trudnością i czasem działania.

Wybór ruchu

- Naturalny wybór: ruch do najlepiej ocenianej sytuacji
- Inna opcja: ruch do sytuacji, w której byliśmy najwięcej razy

MCTS. Dodatkowe uwagi

- Rozgrywka nie musi być prostym losowaniem, p-stwo ruchu może zależeć od jego (szybkiej!) oceny.
- Im więcej symulacji, tym lepsza gra – precyzyjne sterowanie trudnością i czasem działania.

Wybór ruchu

- Naturalny wybór: ruch do najlepiej ocenianej sytuacji
- **Lepsza opcja: ruch do sytuacji, w której byliśmy najwięcej razy**

Komentarz do wyboru ruchu

- W pewnym sensie opcje są podobne: UCB też raczej wybiera dobre ruchy (eksploatacja!)
- Wybierając częstą sytuację, uwzględniamy wiarygodność szacunków
- Pojedyncza bardzo korzystna partia zmienia stosunkowo niewiele

Jeszcze o rozgrywce i wyborze węzła w MCTS

- Ciekawa idea: **all-moves-as-first**: w danej sytuacji na planszy szacujemy jakość ruchów widzianych (w symulacjach, w $\alpha\beta$-search też by się dało to zastosować) niezależnie od tego, w którym momencie się zdarzyły
- Motywacja: w tej sytuacji **zawsze** jak ruszę hetmanem na B5 to wygrywam
- Możemy liczyć wartość ruchu jako średni wynik rozgrywki, w której ten ruch był wykonany.
- **Uwaga**: nie $Q(s, a)$, ale $Q(a)$! (ta wartość nie zależy od konkretnego momentu, w którym ruch został wykonany)

Więcej szczegółów w pracy S.Gelly, D.Silver, *Monte-Carlo Tree Search and Rapid Action Value Estimation in Computer Go*

Stosowalność MCTS

- Nie tylko do gier!
- Można stosować do *poważnych* zadań, związanych z przeszukiwaniem (bez oponenta)
  - Na przykład do rozwiązywania więzów (pewnie szczegóły na ćwiczeniach)