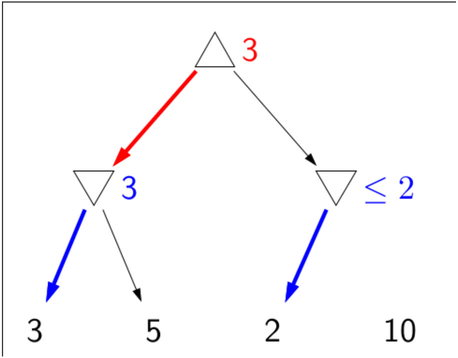
Symulacje w grach. Podstawy teorii gier

Paweł Rychlikowski

Instytut Informatyki UWr

30 kwietnia 2019

Obcinanie fragmentów drzew



Źródło: CS221, Liang i Ermon

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

Obcinanie fragmentów drzew

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

Algorytm A-B

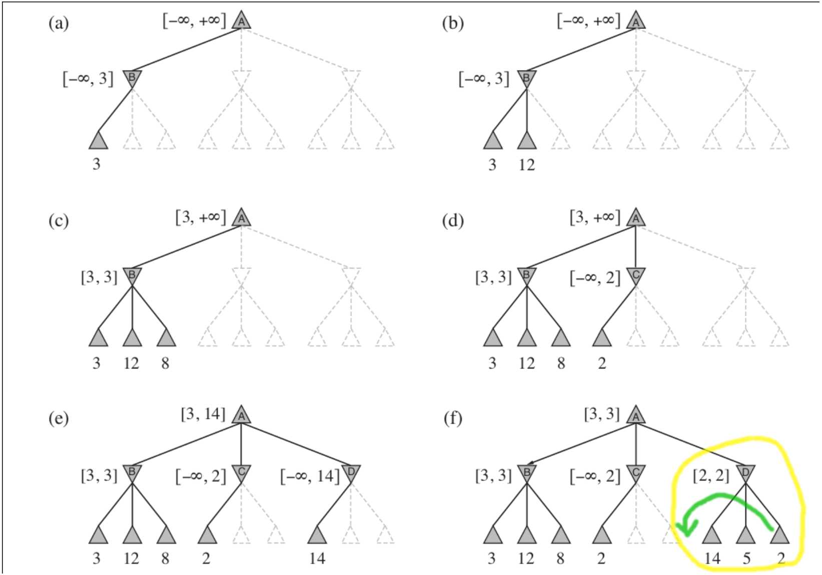
```
def max_value(state, alpha, beta):
    if terminal(state): return utility(state)
    value = -infinity

    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

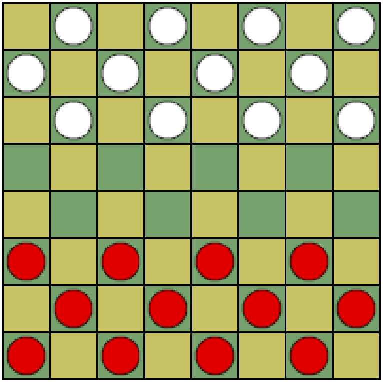
- Efektywność obcięć zależy od porządku węzłów.
- Dla losowej kolejności mamy czas działania $O(b^{2\times 0.75d})$ (czyli efektywne zmniejszenie głębokości do $\frac{3}{4}$)

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

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. 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.

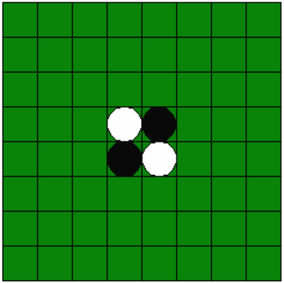
W programie Samuela użyty był wariant pierwszy. Program starał się tak modyfikować parametry funkcji uczącej, żeby możliwie przypominała **minimax** dla głębokości 3 z bardzo prostą funkcją oceniającą (liczącą bierki).

Reversi

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

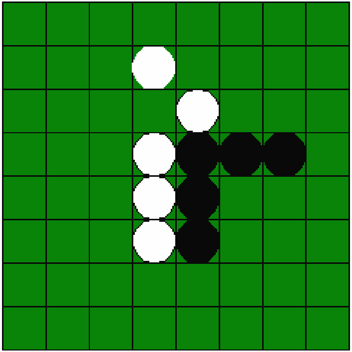
Nadaje się dość dobrze do prezentacji pewnych idei związanych z grami: uczenia i Monte Carlo Tree Search.

Reversi. Zasady

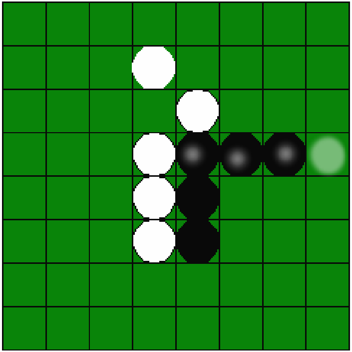


- Zaczynamy od powyższej pozycji.
- Gracze na zmianę dokładają pionki.
- Każdy ruch musi byś biciem, czyli okrążeniem pionów przeciwnkika w wierszu, kolumnie lub linii diagonalnej.
- Zbite pionki zmieniają kolor (możliwe jest bicie na więcej niż 1 linii).
- Wygrywa ten, kto pod koniec ma więcej pionków.

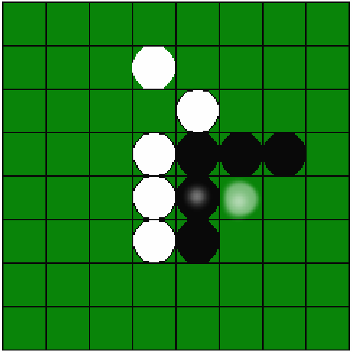
Reversi. Ruchy



Ruch przypada na białego.

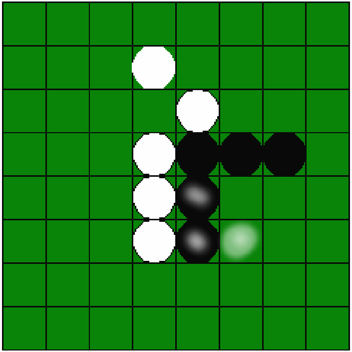


Bicie w poziomie



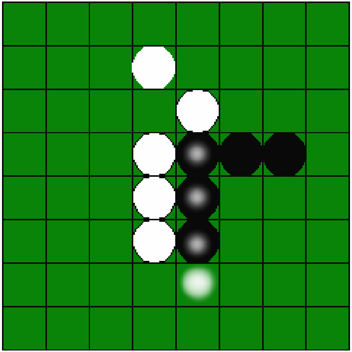
Bicie w poziomie

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_orginal.py

Przykładowa gra. Wnioski

Spróbujmy zanalizować, co się stało.

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

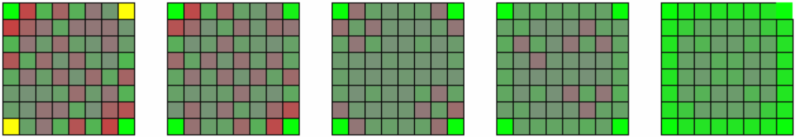
Cel

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, cześciej przegrywamy niż wygrywamy.



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

Eksploracja i eksploatacja

- 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.
- Bardzo wyraźnie widać dylemat eksploracja vs eksploatacja.

Wieloręki bandyta. Przykładowe strategie

- **Zachłanna**: każda rączka po razie, a następnie... ta która dała najlepszy wynik.
- ε**-zachłanna**: rzucamy monetą. Z $p = \varepsilon$ wykonujemy ruch losową rączką, z $p = 1 - \varepsilon$ – wykonujemuy ruch rączką, która ma naljepszy 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

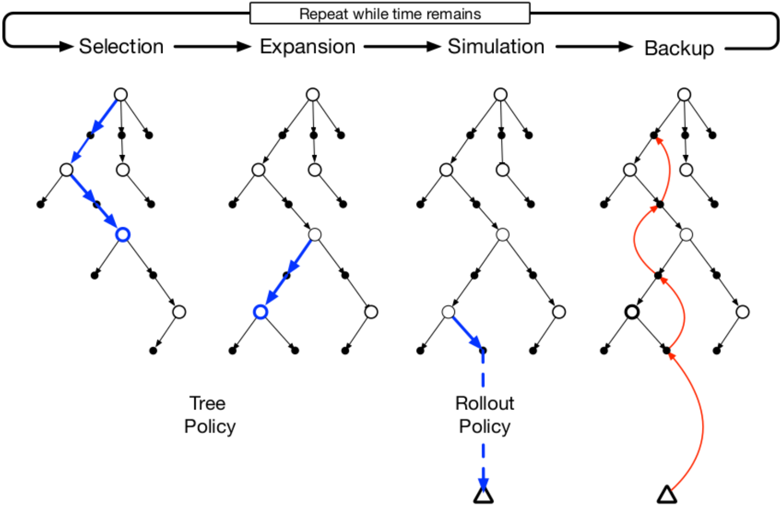
Główne idee

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. Rozwijamy wybrany węzeł (UCB) 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



MCTS. Dodatkowe uwagi

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

Gry z jedną turą

- Powiemy sobie trochę o grach z jedną turą
- Ale takich, w których gracze podejmują swoje decyzje jednocześnie

Rozważamy gry z sumą zerową.

Gra w zgadywanie (Morra 2)

- Mamy dwóch graczy:
  A) Zgadywacz
  B) Zmyłek

  którzy na sygnał pokazują 1 lub 2 palce.
- Jeżeli Zgadywacz nie zgadnie (pokazał coś innego niż Zmyłek), daje Zmyłkowi 3 dolary.
- Jeżeli Zgadywacz zgadnie, to dostaje od Zmyłka:
  - jak pokazali 1 palec, to 2 dolary
  - jak pokazali 2 palce, to 4 dolary

Pytanie

Jak grać w tę grę? (prśba o podanie wstępnych intuicji)

Macierz wypłat

Definicja

Taką grę zadajemy za pomocą macierzy wypłat, w której $V_{a,b}$ jest wynikiem gry z punktu widzenia pierwszego gracza.

Nasza gra:

Zg/Zm	1 palec	2 palce
1 palec	2	-3
2 palce	-3	4

Strategie

- Czysta strategia: zawsze akcja a
- Mieszana strategia: rozkład prawdopobieństwa na akcjach

Proste fakty

- Jak Zmyłek będzie grał cały czas to samo, to Zgadywacz wygra każdą turę (i odwrotnie)
- Muszą zatem stosować strategie mieszane, ale jakie?

Wartość gry

Definicja

Wartość gry dla dwóch strategii graczy jest równa:

$$V(\pi_A, \pi_B) = \sum_{a,b} \pi_A(a)\pi_B(b)V(a,b)$$

Przykładowo: Zgadywacz zawsze zgaduje 1, Zmyłek wybiera akcję losowo z prawdopodobieństwem 0.5.

Wynik: $-\frac{1}{2}$ (tak samo często zyskuje 2 jak traci 3 dolary)

Strategia mieszana vs czysta

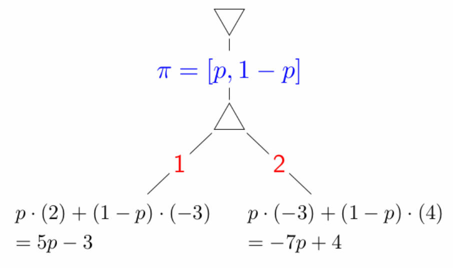
Uwaga

Jeżeli gracz A zapowie, że będzie grał strategią mieszaną (i ją poda), wówczas gracz B może grać strategią czystą (i osiągnie optymalny wynik).

Dlaczego?

Znalezienie optymalnej strategii

Zaczyna gracz B – Zmyłek.
Wybiera strategię mieszaną z parametrem p



Wartość takiej gry to

$$\min_{p \in [0,1]} (\max(5p - 3, -7p + 4))$$

Zauważmy, dla jakich p wygrywa lewe, dla jakich prawe i co z tego wynika.

Znalezienie optymalnej strategii (2)

- W powyższej grze, Zmyłek osiągnie najlepszy wynik, gdy przyjmie $p = \frac{7}{12}$, wynik ten to $-\frac{1}{12}$
- Ok, on zaczynał, miał trudniej – a gdyby zaczynał Zgadywacz? I podał swoją strategię mieszaną?

Wynik gry

Wynik jest dokładnie taki sam, czyli $-\frac{1}{12}$!

Twierdzenie von Neumana

Twierdzenie, von Neuman, 1928

Dla każdej jednoczesnej gry dwuosobowej o sumie zerowej ze skończoną liczbą akcji mamy:

$$\max_{\pi_A} \min_{\pi_B} V(\pi_A, \pi_B) = \min_{\pi_B} \max_{\pi_A} V(\pi_A, \pi_B)$$

dla dowolnych mieszanych polityk π_A, π_B.

- Można ujawnić swoją politykę optymalną!
- **Dowód**: pomijamy, programowanie liniowe, przedmiot J.B.
- Algorytm: programowanie liniowe