O grach (część 2)

Paweł Rychlikowski

Instytut Informatyki UWr

8 kwietnia 2021

Przykładowa gra. Przypomnienie

- Gracz **A**, który chce skończyć z jak największą liczbą, wybiera jeden z trzech zbiorów:
  1. $\{-50, 50\}$
  2. $\{1, 3\}$
  3. $\{-5, 15\}$
- Następnie gracz **B** wybiera liczbę z tego zbioru.

Składniki

Stany gry (początkowy, 3 po pierwszym ruchu, 6 po drugim), ruchy i mechanika, wypłaty w stanach końcowych.

Uwaga

W grach o sumie zerowej/stałej
wypłata drugiego gracza = K - wypłata pierwszego (często K=0)

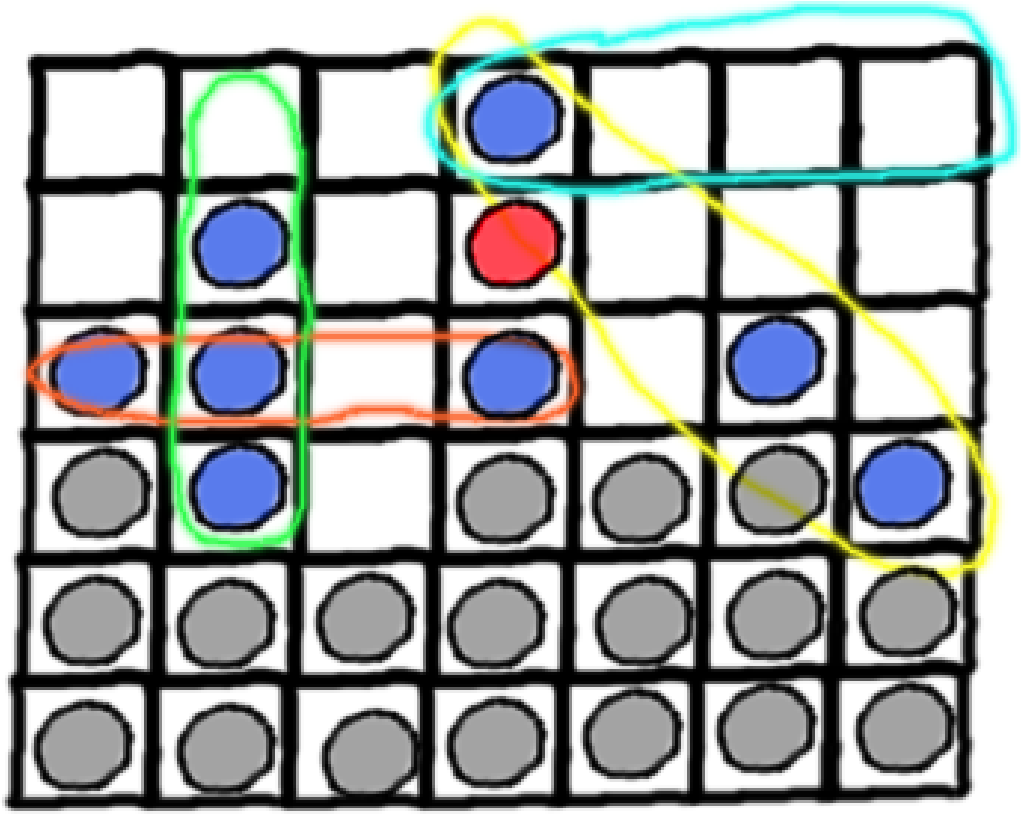
Connect 4. Inna przykładowa gra

- Prosty, a zarazem grywalny wariant kółka i krzyżyka
- Dodatkowe elementy: mamy ciążenie i piony spadają, gramy do 4 w wierszu, kolumnie lub na przekątnej

Connect 4. Inna przykładowa gra

- Prosty, a zarazem grywalny wariant kółka i krzyżyka
- Dodatkowe elementy: mamy ciążenie i piony spadają, gramy do 4 w wierszu, kolumnie lub na przekątnej

Co to znaczy wzorzec w Connect 4?



- Analizujemy wszystkie czwórki pól (w każdej bowiem może się zdarzyć układ wygrywający)
- Czwórki, w których są pionki obu kolorów pomijamy
- Wyznaczamy wagę 1-ek, dwójek, trójek (być może zależnie od kierunków)

Wzorce w Connect 4

- Możliwe są większe wzorce, uwzględniające szerszy kontekst
- możliwe jest również **uczenie** większych wzorców. Na przykład za pomocą splotowych sieci neuronowych (CNN).

Uwaga

Takie sieci działały w AlphaGo.

Ewaluacja z wzorców

- Funkcja oceny może być ważoną sumą zaobserwowanych wzorców.
- Wzór:

$$\sum_i w_i p_i$$

(w_i – waga i-tego wzorca, p_i – ile razy ten wzorzec występuje na planszy)

- Niektóre wagi są dodatnie (mój dobry wzorzec, słabe ustawienie oponenta), inne ujemne.

Drobna uwaga o ewolucji. Jak wyznaczyć parametry funkcji oceniającej?

- Istnieje pokusa, żeby zastosować algorytmy ewolucyjne (bo zadanie przypomina ewolucje, w której osobniki toczą ze sobą walkę).
- **Problem:** Jak wyznaczyć funkcję celu?
  a) Rozgrywać turnieje, przystosowaniem jest średni wynik.
  b) Wybrać grupę przeciwników (stałą), przystosowaniem X-a będzie średni wynik z tymi przeciwnikami.

Uwaga

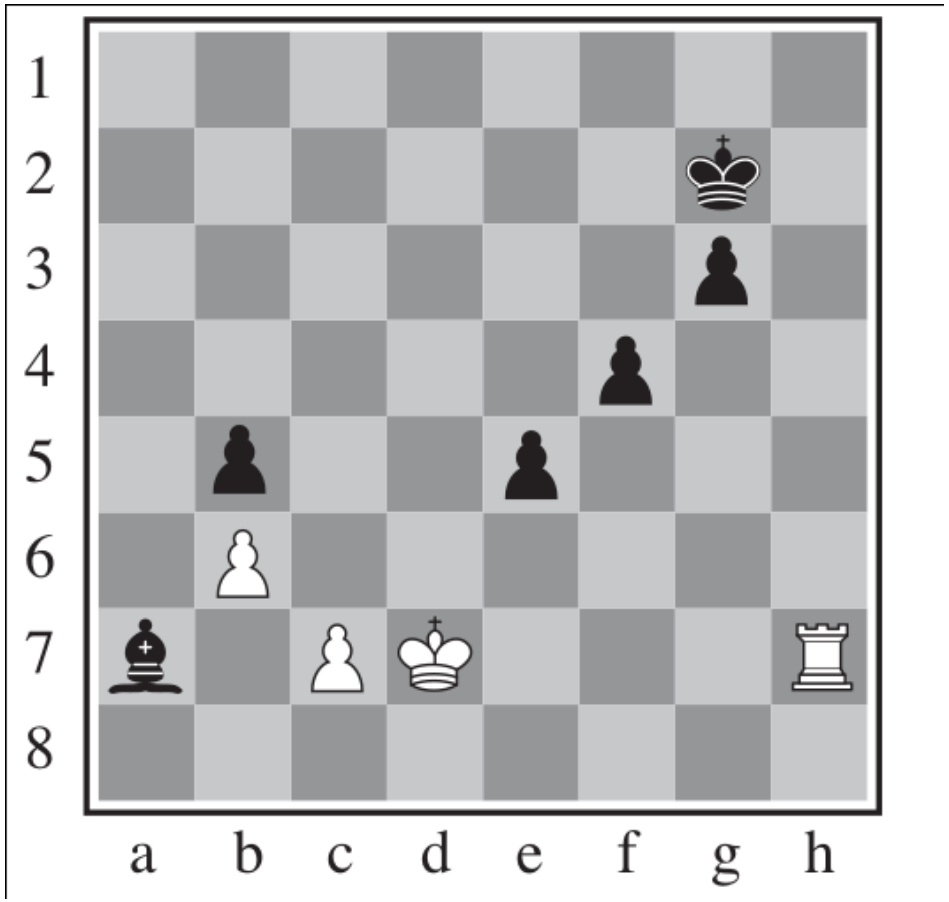
Opcja pełnej ewolucji trochę niebezpieczna, często łączy się oba warianty.

Przerywanie przeszukiwania

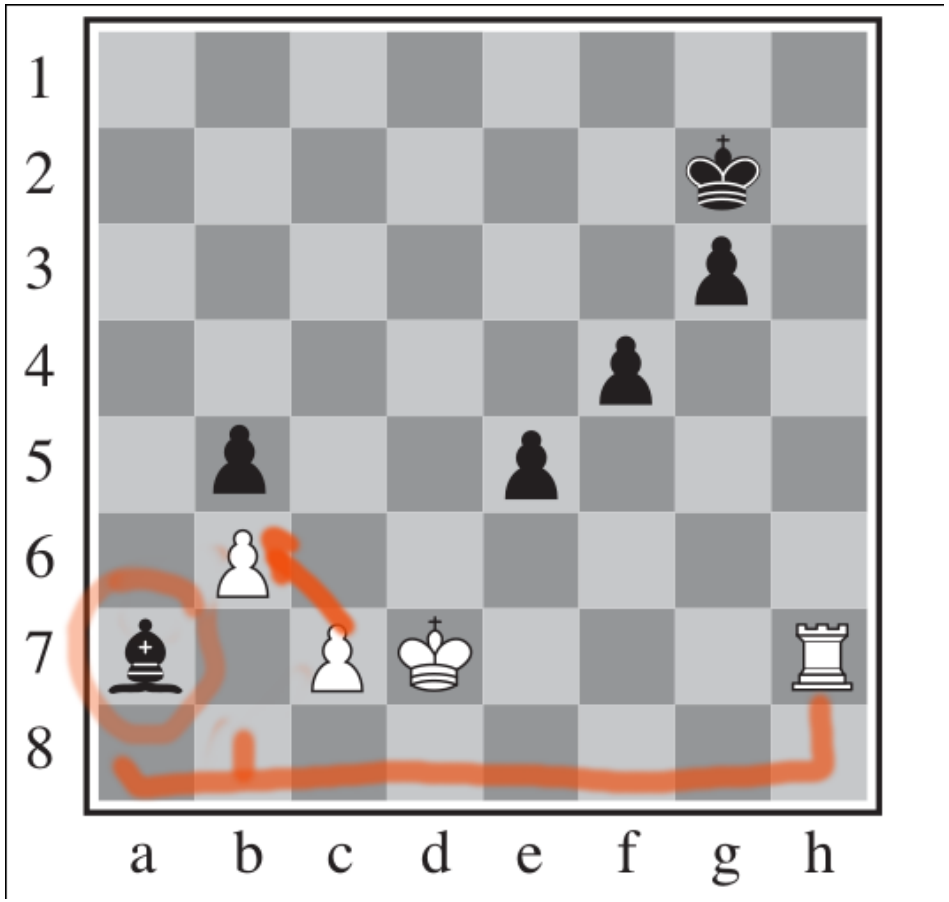
Drugi metaparametr funkcji obliczającej wartość planszy.

- Są dwa problemy związane z przerywaniem przeszukiwania:
  1. Przerwanie w niestabilnej sytuacji (na przykład w środku wymiany hetmanów)
  2. Tzw. efekt horyzontu (czyli widzimy, że coś się zdarzy, ale w odległej perspektywie)

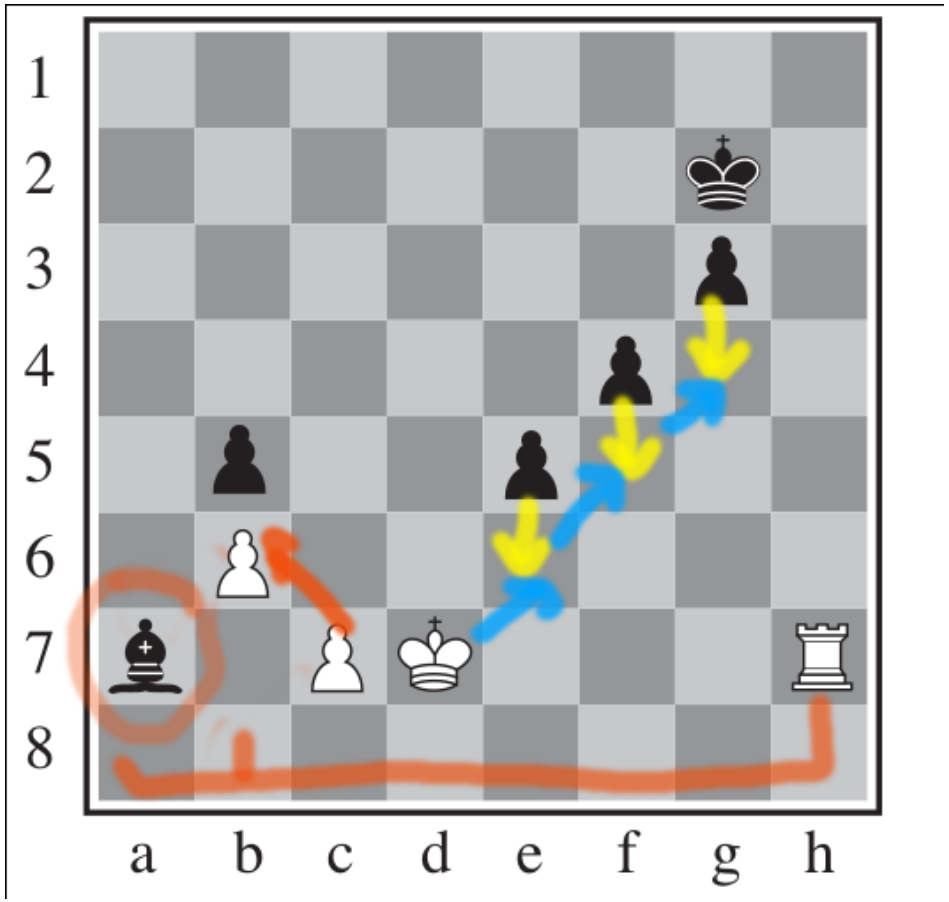
Efekt horyzontu (zła sytuacja czarnego gońca)



Efekt horyzontu (zła sytuacja czarnego gońca)



Efekt horyzontu (zła sytuacja czarnego gońca)



Kończenie przeszukiwań w praktyce

- Nieprzerywanie, jeżeli przeciwnik ma bicie.
- Ogólniej: powyżej jakiejś głębokości rozważamy tylko ruchy mocno zmieniające sytuację

Definicja

W **przeszukiwaniu z bezruchem** (quiescence search) możemy skończyć poszukiwanie **tylko** gdy sytuacja jest statyczna.

Kończenie przeszukiwań w praktyce

- Można też stosować jakąś wersję *local beam search* (od któregoś momentu ograniczając mocno rozgałęzienie drzewa)
- Rozważa się warunek singular extension, czyli istnienie jednego ruchu, który jest wyraźnie (na oko) lepszy od innych. Takie ruchy zawsze wykonujemy, zwiększając głębokość, a nie zwiększając rozgałęzienia.

Uwaga

Trochę tak działają ludzie.

Koniec części I

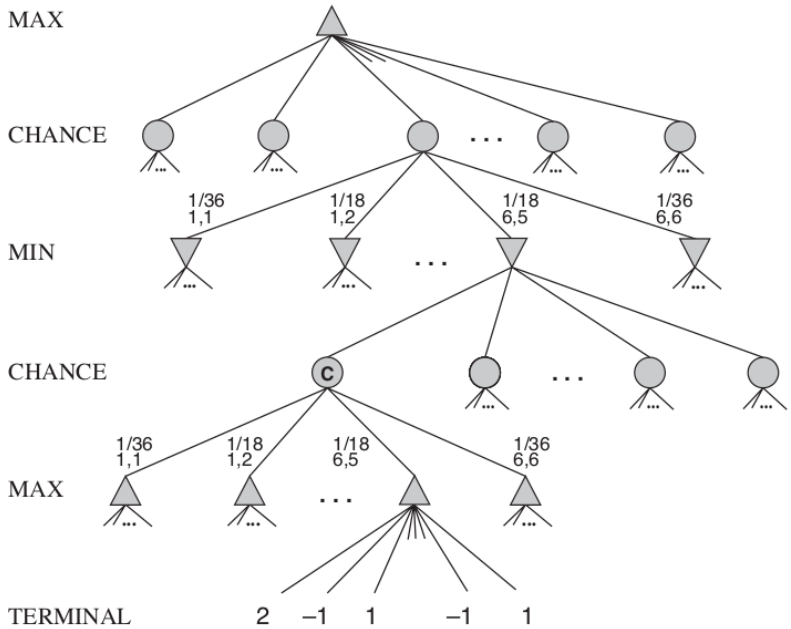
Inne warianty gier

Losowość w grach

- W niektórych grach (i w życiu) mamy element losowy.
- Prosty przykład: szachy z kostką:
  - Przed ruchem wykonujemy rzut kostką, który determinuje czym możemy się ruszyć,
  - 1 -pionek, 2 - skoczek, 3 - goniec, 4 – wieża, 5 – hetman, 6 – król
  - Gramy do zbicia króla.

Losowość w grach

- Wprowadzamy dodatkowe węzły, czyli chance nodes.
- Przykładowe drzewo gry (dla losowania przy użyciu dwóch kości):



Expectimax

- Minimax, do którego dołożono węzły losowe.
- W węzłach losowych mamy wybór wartości oczekiwanej (sumowanie)

```
def emm(state, player):
    if terminal(state): return utility(state)
    if player == MIN:
        return min( emm(result(state, a), next(player)) for a in actions(state))
    if player == MAX:
        return max( emm(result(state, a), next(player)) for a in actions(state))
    if player == CHANCE:
        return sum( P(r) * emm(result(state, r), next(player)) for r in actions(state))
```

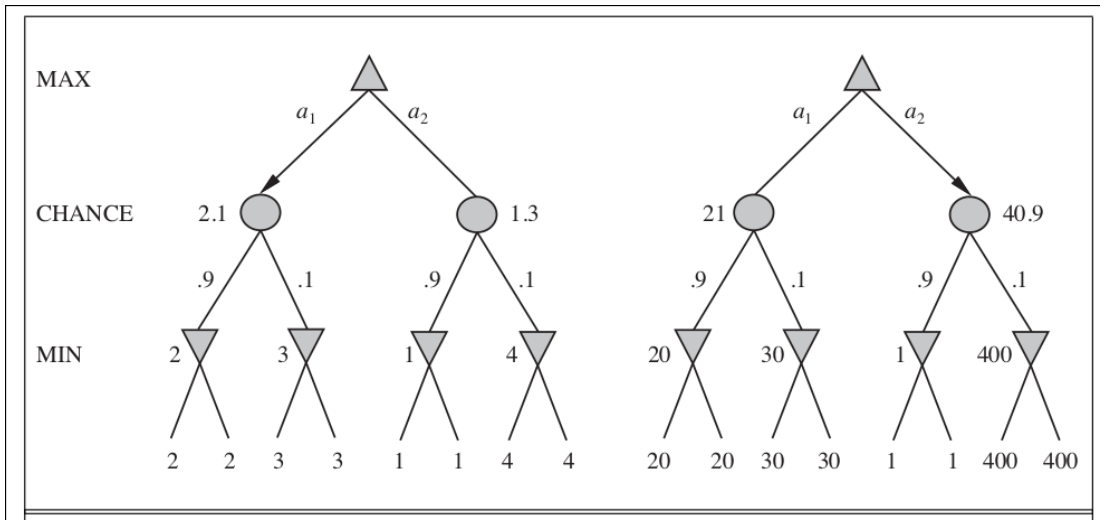
Wartość sytuacji w grach z losowością

Uwaga 1

Dowolne monotoniczne przekształcenie nie zmienia ruchów wybieranych przez minimax!

Uwaga 2

W grach z losowością powyższe zdanie przestaje być prawdziwe.



Monte Carlo Simulation

- Analiza gier z losowością jest nieco trudniejsza.
- Możemy skorzystać z następującej idei:

  Oceniamy sytuację przeprowadzając dużo losowych gier rozpoczynających się w danej sytuacji

- **Uwaga:** dwa rodzaje losowości: jeden związany z węzłami losowymi (dany przez grę), drugi związany z węzłami min/max – zamiast wyliczać ruch wykonujemy ruch losowy.

Uwaga

Monte Carlo Simulation dotyczy nie tylko gier z losowością!

Monte Carlo Simulation

- Zauważmy, że Monte Carlo Simulation jakoś rozwiązuje problem horyzontu (bo symulacje mogą być b. długie)
- Możemy losować ruchy z niejednakowym prawdopodobieństwem (preferując te, które lokalnie wyglądają sensownie)

Uwaga

Bardzo ważnym nie tylko w grach jest algorytm Monte Carlo Tree Search, o którym jeszcze powiemy.

Gry częściowo obserwowalne

- Ciekawe do analizy są gry, w których agenci nie mają pełnej wiedzy o świecie.
- Klasyczne przykłady to gry karciane, ale nie tylko.

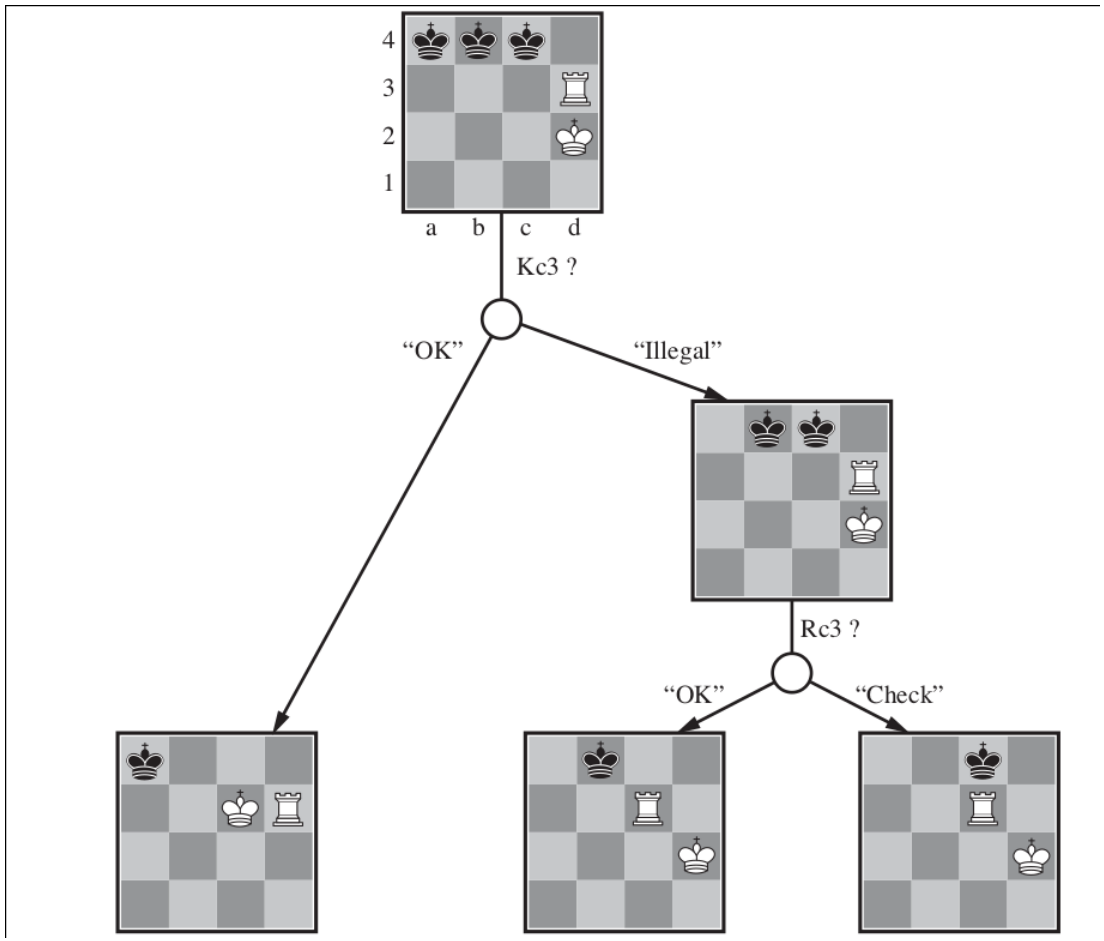
Kriegspiel

- Mamy dwóch graczy, arbitra i 3 szachownice.
- Gracze widzą na szachownicy swoje pionki, mogą tworzyć też hipotezy o bierkach przeciwnika.
- Arbiter zna położenie wszystkich figur i udziela graczom pewnych (skąpych) informacji.
  a) przede wszystkim ocenia, czy ruch jest możliwy (komunikacja osobista, dobry ruch jest od razu wykonywany, w przypadku złego, gracz proponuje kolejny, aż do skutku)
  b) odpowiada na pytanie: „czy ja (gracz) mam jakieś bicie?"
  c) informuje obu graczy, że „na polu X zbito bierkę" (nie podając jaka bierka jest zbita, a jaka biła)
  d) Mówi o szachu (do ubu graczy), dodając, że zagrożenie jest w wierszu, kolumnie, przekątnej lub przez skoczka
- Tak poza tym, to całkiem normalne szachy.

Podobno ludzie radzą sobie z tą grą całkiem nieźle...

Końcówka w Kriegspiel

Przykładowa końcówka, gracz biały dowiedział się, że czarnemu został tylko król i jest on na jednym z 3 pól.



Uwaga 1

W stanie gry powinniśmy umieścić możliwe ustawienia bierek przeciwnika

Trochę jak z komandosem...

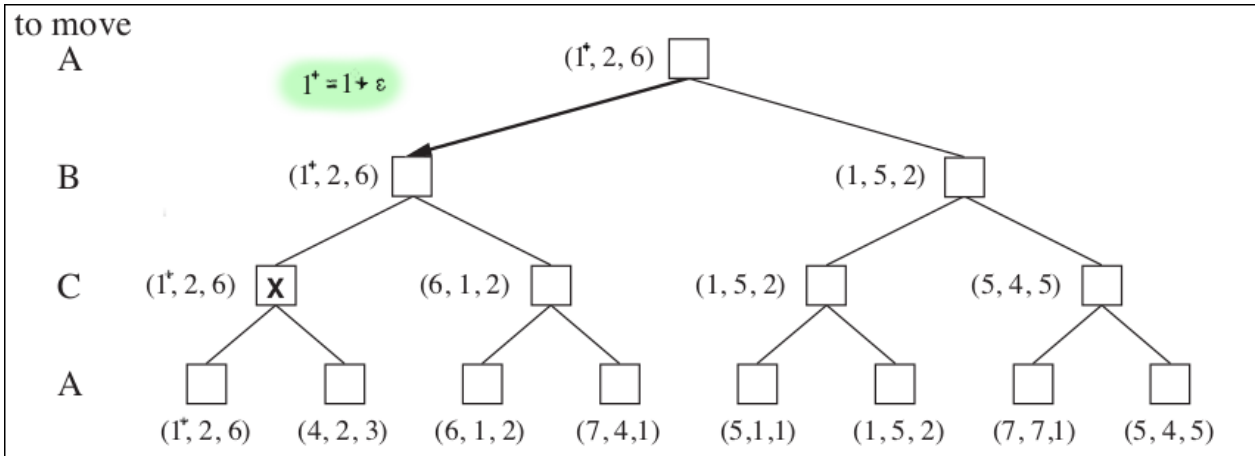
Pytanie

A jak grać w brydża, bądź inną grę karcianą?

Idea (do rozwinięcia na ćwiczeniach)

losowanie układu kart i gra w otwarte karty dla wylosowanego układu, czynności powtarzamy wiele razy

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.

Koniec części II

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

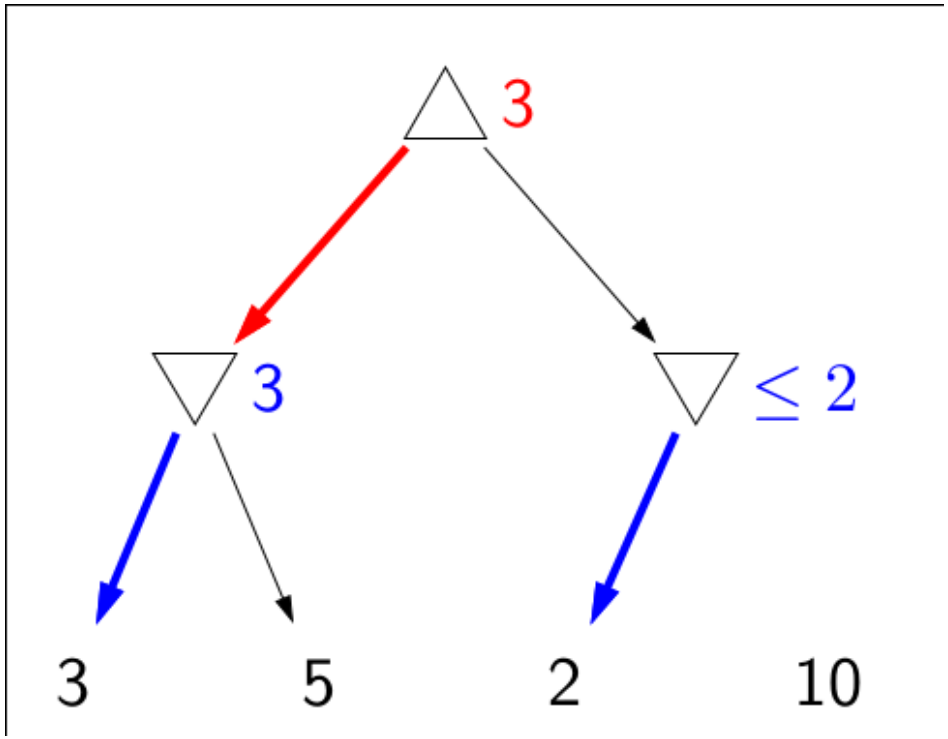
Uwaga

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



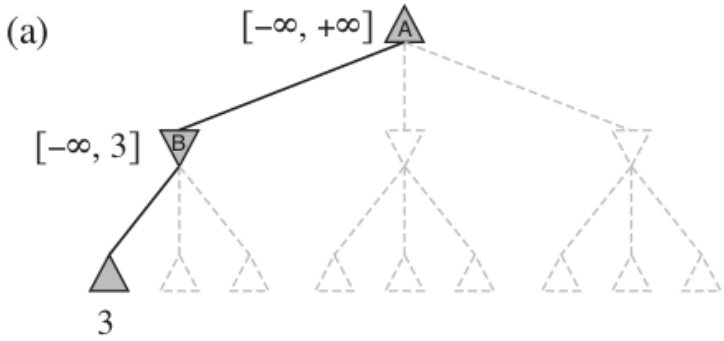
Źródło: CS221, Liang i Ermon

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

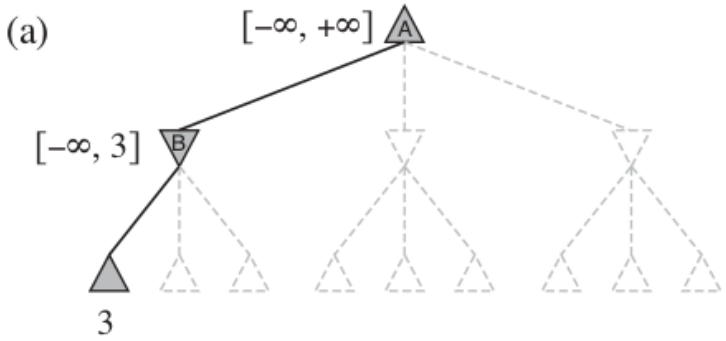
Obcinanie fragmentów drzew

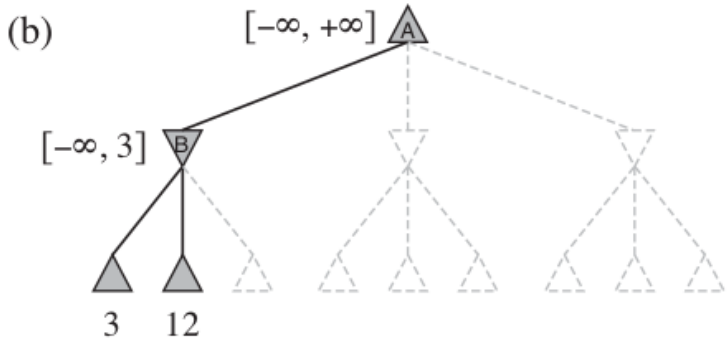
- Jeżeli możemy udowodnić, że w jakimś poddrzewie nie ma optymalnej wartości, to możemy pominąć to poddrzewo.
- Będziemy pamiętać:
  - α – dolne ograniczenie dla węzłów MAX ($\geq \alpha$)
  - β – górne ograniczenie dla węzłow MIN ($\leq \beta$)

Alfa-Beta w akcji

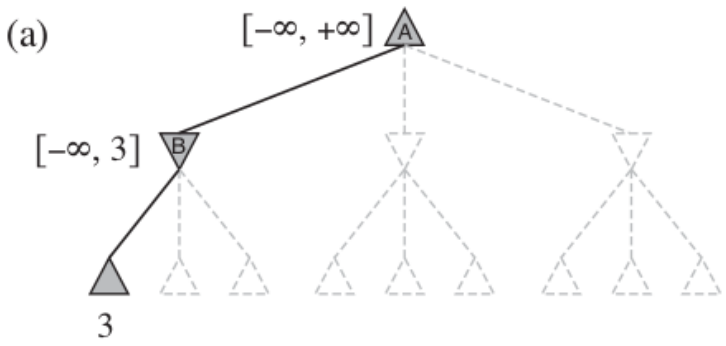


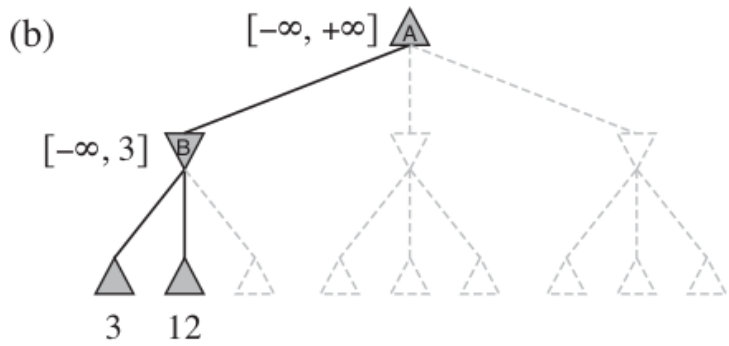
Alfa-Beta w akcji

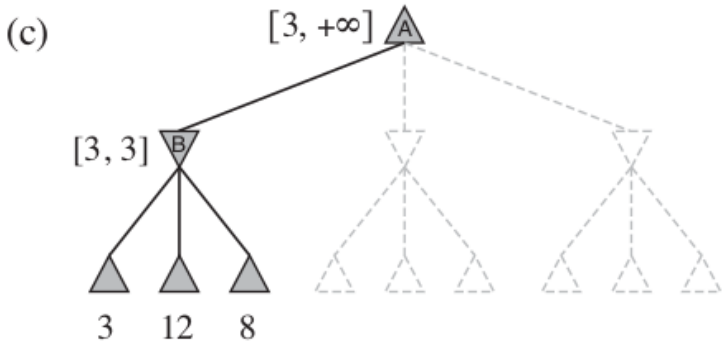




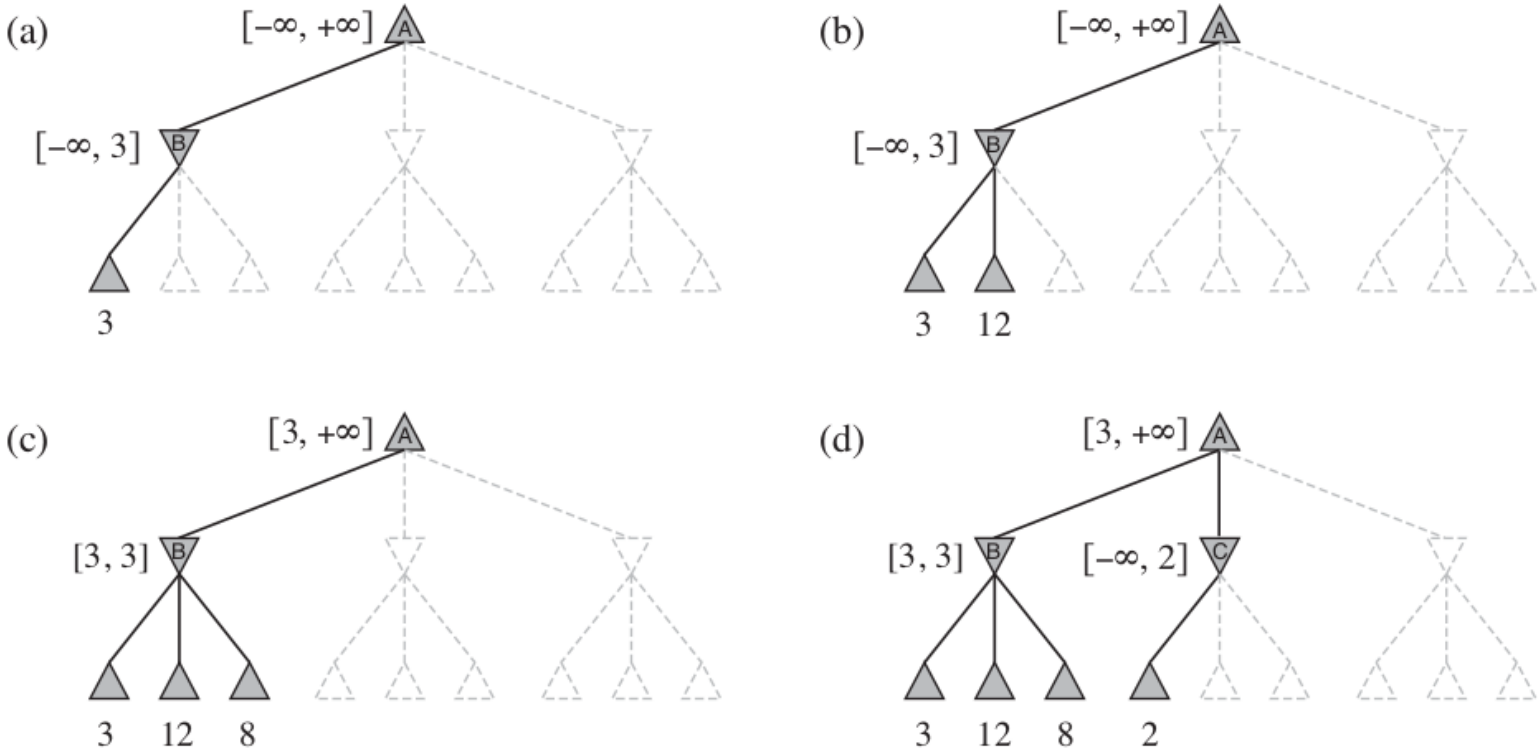
Alfa-Beta w akcji



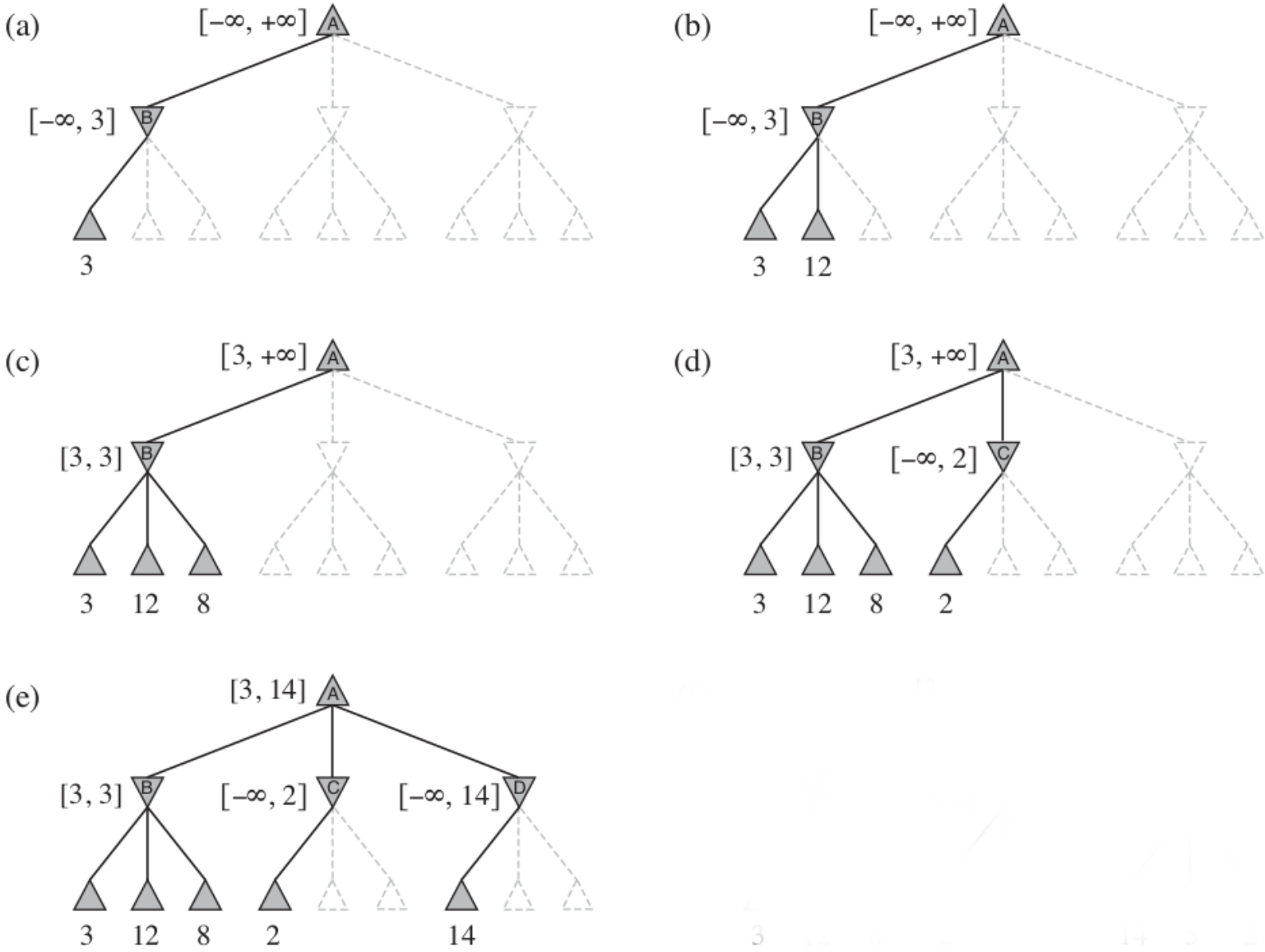




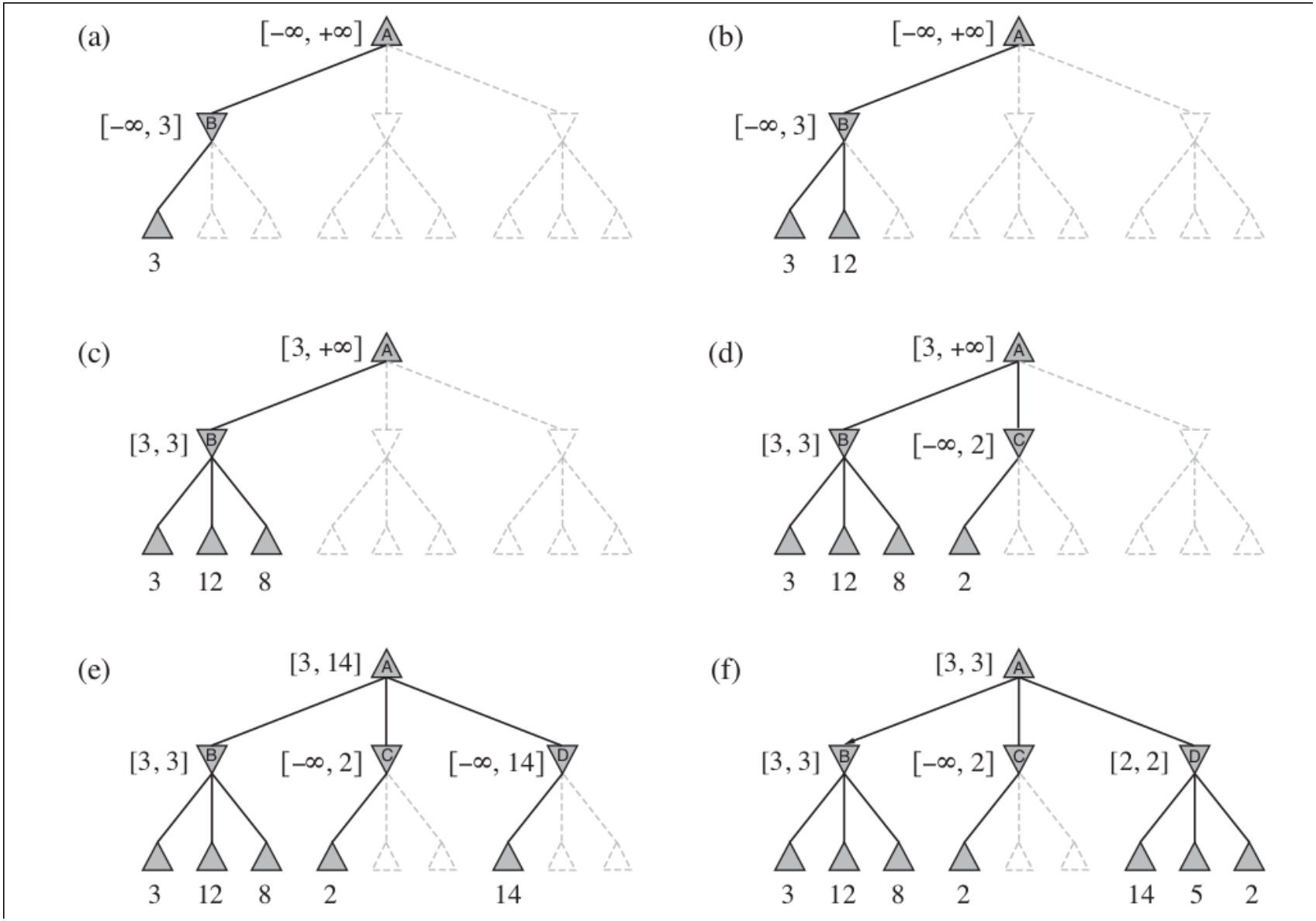
Alfa-Beta w akcji



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

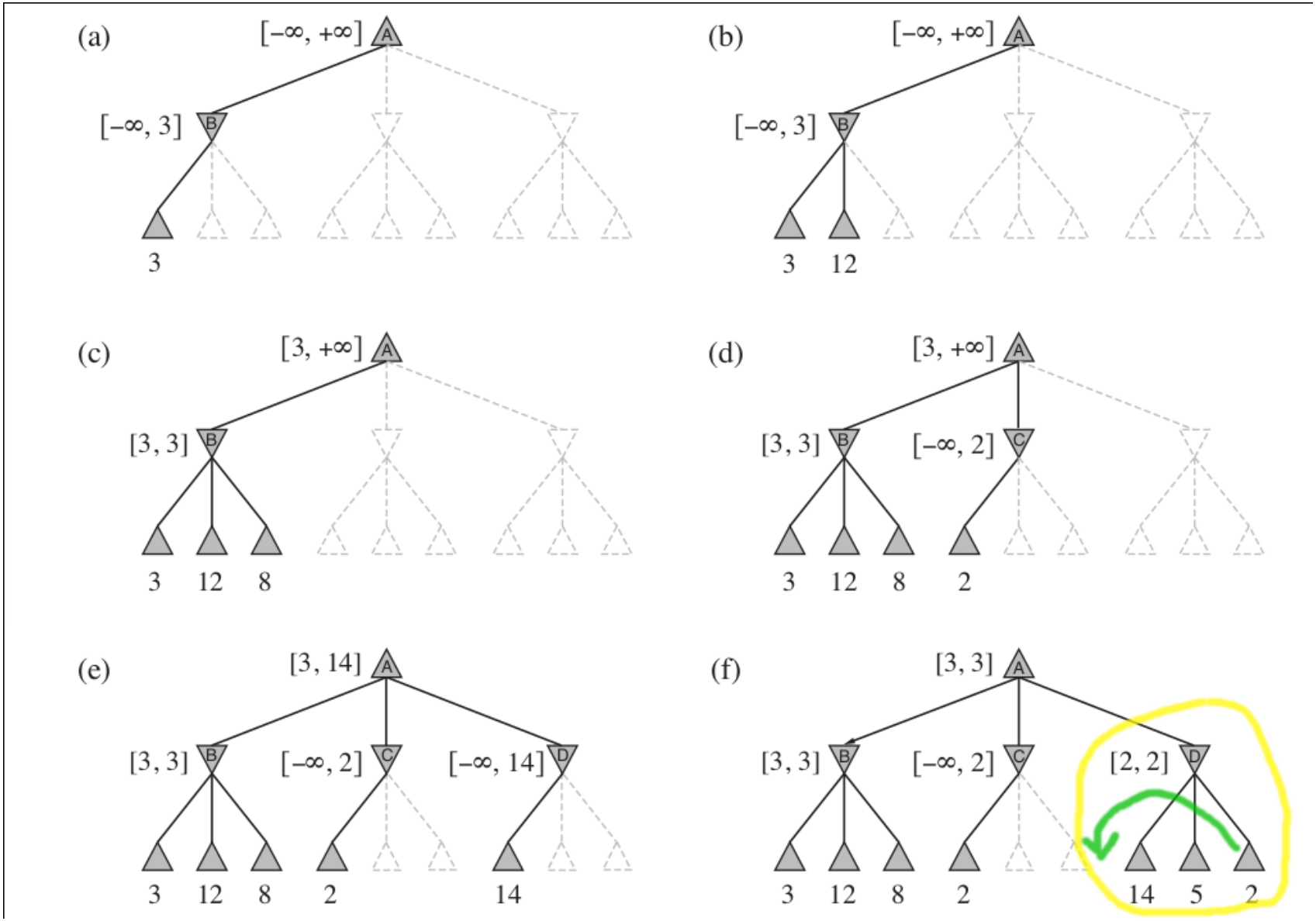
- Efektywność obcięć zależy od porządku węzłów.
- Dla losowej kolejności mamy czas działania $O(b^{2\times 0.75d})$ (czyli efektywne zmniejszenie głębokości do $\frac{3}{4}$)

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

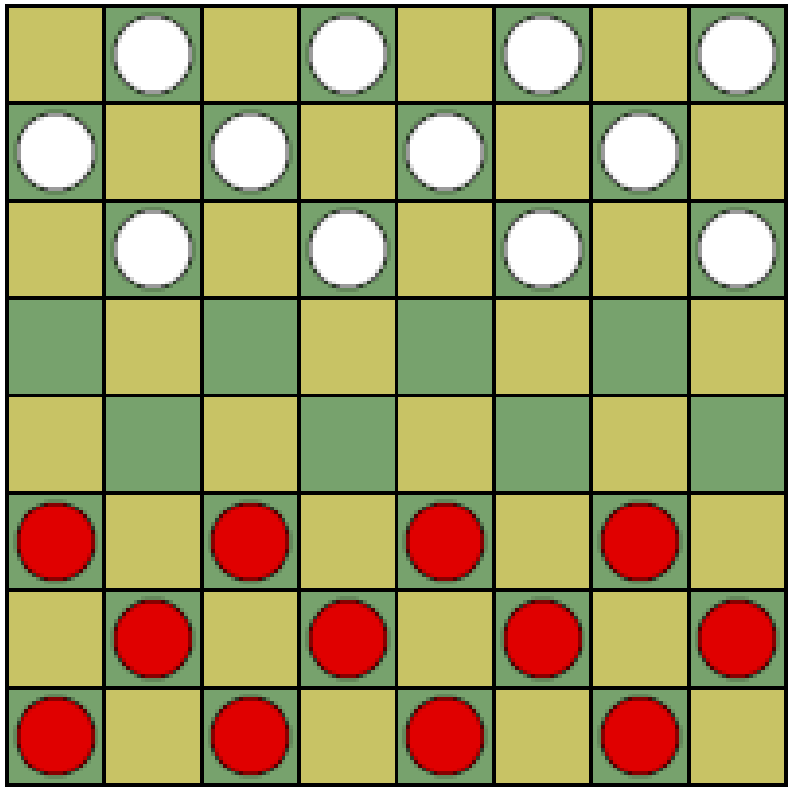
Uwaga

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).