SID – Wykład 4
Gry

Dominik Ślęzak

Wydział Matematyki, Informatyki i Mechaniki UW
slezak@mimuw.edu.pl

Gry a problemy przeszukiwania

"Nieprzewidywalny" przeciwnik $\Rightarrow$ rozwiązanie jest strategią specyfikującą posunięcie dla każdej możliwej odpowiedzi przeciwnika

Ograniczenia czasowe $\Rightarrow$ mało prawdopodobne znalezienie celu, trzeba aproksymować

Historia:

- Maszyna rozważa różne scenariusze rozgrywki (Babbage, 1846)
- Algorytmy dla gier z pełną informacją (Zermelo, 1912; Von Neumann, 1944)
- Backward-induction algorithm (von Neumann, Morgenstern, 1944)
- Skończony horyzont, aproksymacyjna ocena stanu gry (Zuse, 1945; Wiener, 1948; Shannon, 1950)
- Program grający w szachy (Turing, 1951)
- Zastosowanie uczenia maszynowego do poprawy trafności oceny stanu gry (Samuel, 1952–57)
- Odcięcia umożliwiające głębsze przeszukiwanie (McCarthy, 1956)

Rodzaje gier

	deterministyczne	**niedeterministyczne**
Pełna informacja	szachy, warcaby, go	backgammon, monopoly
Niepełna informacja		bridge, poker, scrabble

Gra deterministyczna: 2 graczy

Gracze: **MAX** i **MIN**

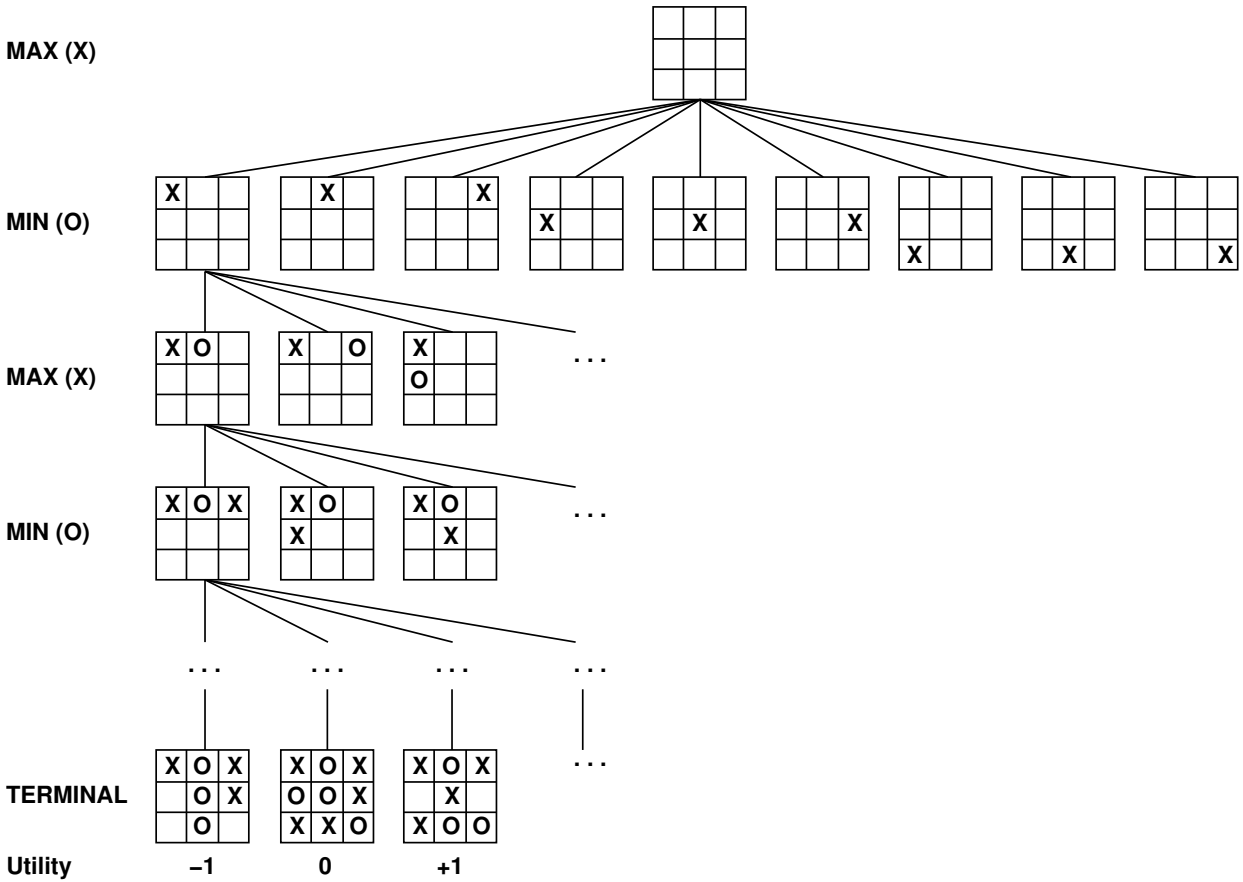
Stan początkowy: stan planszy i wskazanie gracza rozpoczynającego (**MAX**)

Funkcja następnika: zbiór par (posunięcie, stan) opisujących wszystkie dopuszczalne posunięcia z bieżącego stanu

Test końca gry: sprawdza, czy stan gry jest końcowy

Funkcja użyteczności (wypłaty): numeryczna wartość dla stanów końcowych, np. wypłaty dla wygranej, porażki i remisu mogą być równe odpowiednio +1, -1 i 0.

Drzewo gry deterministycznej: 2 graczy



Strategia minimax: algorytm

Dla gier deterministycznych z pełną informacją.

Pomysł: wybiera ruch zapewniający największą wypłatę tzn. największą wartość minimax (funkcja MINIMAX-VALUE) przy założeniu, że przeciwnik gra optymalnie.

function MINIMAX-DECISION(*state*) **returns** an *action*
    action, *state* ← the *a*, *s* in SUCCESSORS(*state*) such that MINIMAX-VALUE(*s*) is maximized
    return *action*
end function

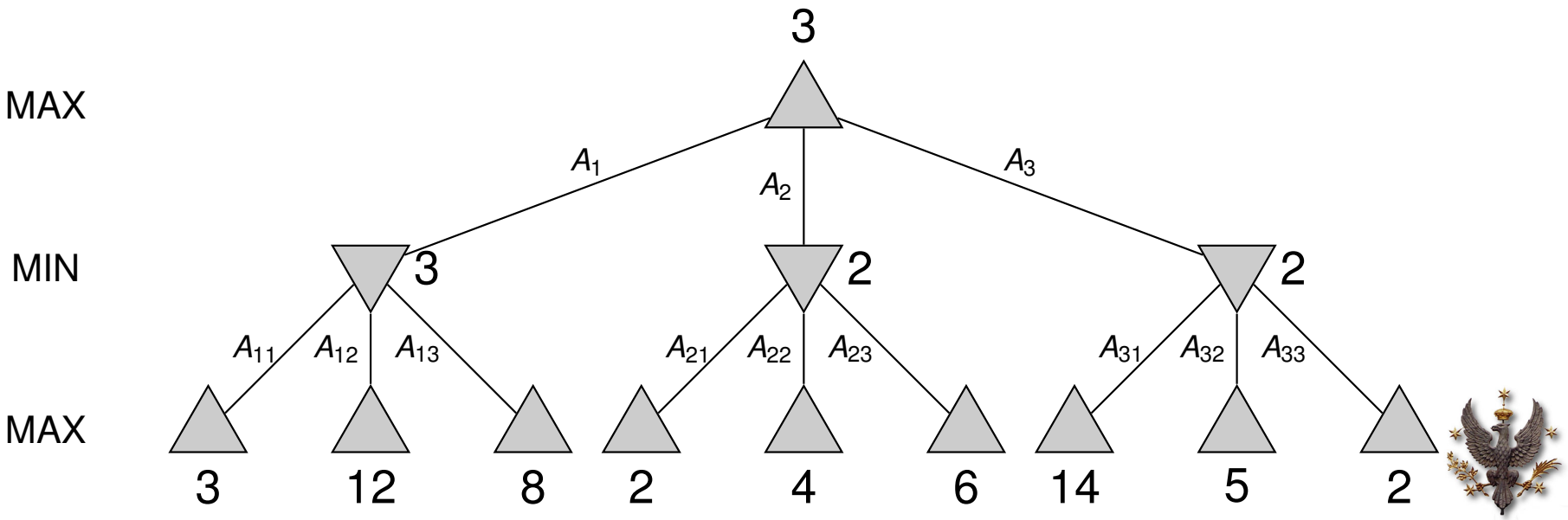
---

function MINIMAX-VALUE(*state*) **returns** a *utility* value
    if TERMINAL-TEST(*state*) **then**
        return UTILITY(*state*)
    else if MAX is to move in *state* **then**
        return the highest MINIMAX-VALUE of SUCCESSORS(*state*)
    else
        return the lowest MINIMAX-VALUE of SUCCESSORS(*state*)
    end if
end function

Strategia minimax: przykład

Gracz **MAX** maksymalizuje funkcję wypłaty (węzły △) $\Rightarrow$ wybiera ruch w lewą gałąź drzewa

Gracz **MIN** minimalizuje funkcję wypłaty (węzły ▽) $\Rightarrow$ wybiera ruch do lewego liścia poddrzewa



Strategia minimax: własności

- Użyteczność: Gry deterministyczne z pełną informacją z dowolną liczbą graczy
- Pełność: Tak, jeśli drzewo przeszukiwań jest skończone
  Gry z nieskończonym drzewem przeszukiwań mogą mieć strategie skończone!
- Optymalność: Tak, jeśli przeciwnik gra optymalnie. W ogólności nieoptymalne.
- Złożoność czasowa: $O(b^m)$, gdzie

  b - maksymalne rozgałęzienie drzewa przeszukiwań
  m - maksymalna głębokość drzewa przeszukiwań
- Złożoność pamięciowa: $O(bm)$ (przeszukiwanie wgłąb)

Dla szachów, $b \approx 35$, $m \approx 100$ dla "sensownych" rozgrywek $\Rightarrow$ dokładne rozwiązanie zupełnie nieosiągalne.

Strategia minimax z odcięciem

Problem: brak czasu na pełne przeszukanie przestrzeni stanów, np. 100 sekund na posunięcie, szybkość 10^4 węzłów/sek $\Rightarrow$ 10^6 węzłów na ruch

Rozwiązanie: przeszukiwanie z odcięciem ograniczającym głębokość przeszukiwania

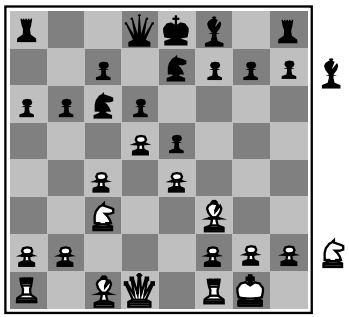
Strategia minimax z odcięciem: algorytm

```
function MINIMAX-DECISION(state) returns an action
    action, state ← the a, s in SUCCESSORS(state) such that MINIMAX-VALUE(s) is maximized
    return action
end function
```

```
function MINIMAX-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then
        return UTILITY(state)
    else if MAX is to move in state then
        return the highest MINIMAX-VALUE of SUCCESSORS(state)
    else
        return the lowest MINIMAX-VALUE of SUCCESSORS(state)
    end if
end function
```

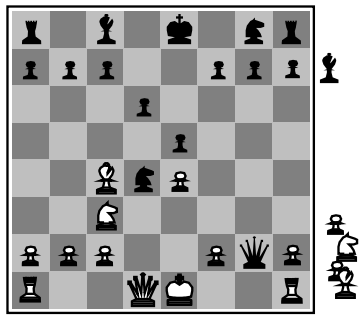
Funkcja oceny UTILITY szacuje wypłatę dla danego stanu gry = rzeczywistej wypłacie dla stanów końcowych

Funkcja oceny: przykład



Black to move

White slightly better



White to move

Black winning

np.

Dla szachów, przeważnie liniowa ważona suma cech

$$UTILITY(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$$

$f_1(s)$ =(liczba białych hetmanów) − (liczba czarnych hetmanów)

$f_2(s)$ =(najmniejsza odległość od linii przemiany dla pionów czarnych)

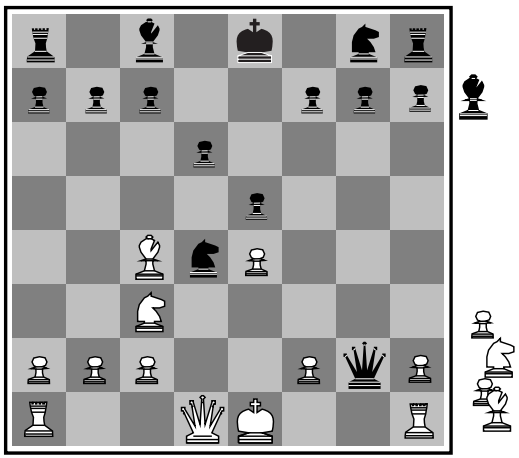
− (najmniejsza odległość od linii przemiany dla pionów białych)

$f_3(s)$ =(1 jeśli była wykonana lub nadal jest możliwa roszada białych, w przeciwnym przypadku 0)

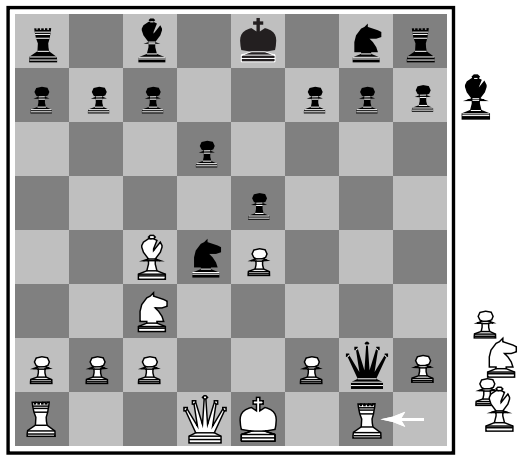
− (1 jeśli była wykonana lub nadal jest możliwa roszada czarnych, w przeciwnym przypadku 0))

Przeszukiwanie stabilne

Problem: Stany mają taką samą wartość oceny (na korzyść czarnych), ale stan z prawej jest niestabilny – kolejny ruch daje dużą zmianę oceny stanu gry (na korzyść białych)



(a) White to move

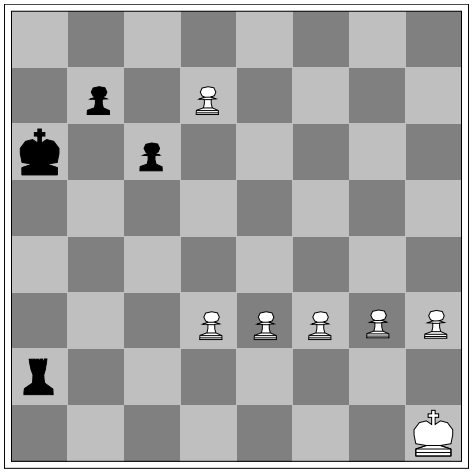


(b) White to move

Rozwiązanie: przeszukiwanie stabilne – stany niestabilne są rozwijane do momentu osiągnięcia stanu stabilnego

Przeszukiwanie z pojedynczym rozwinięciem

Efekt horyzontu: gracz wykonuje ruchy odsuwając nieuniknione posunięcie na korzyść przeciwnika poza horyzont przeszukiwania, np. czarna wieża powtarza szachowanie białego króla



Black to move

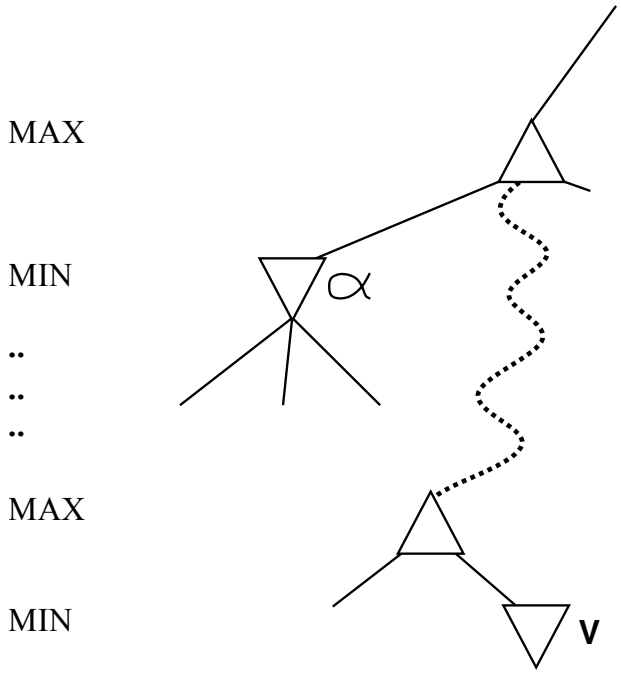
Rozwiązanie: przeszukiwanie z pojedynczym rozwinięciem – algorytm wykonuje pogłębione przeszukiwanie dla wybranych posunięć "wyraźnie lepszych" od pozostałych

Strategia minimax z odcięciem α–β

α jest najlepsza wartością dla MAX poza bieżącą ścieżką przeszukiwania

Jeśli V jest gorsze niż α, MAX nigdy nie wejdzie do tej gałęzi $\Rightarrow$ gałąź z V można odciąć.

β jest definiowane analogicznie dla MIN



Strategia minimax z odcięciem α–β: algorytm

function ALPHA-BETA-SEARCH(*state*) **returns** an *action*
  $v \leftarrow$ MAX-VALUE(*state*, $-\infty$, $+\infty$)
  return the *action* in SUCCESSORS(*state*) with value v
end function

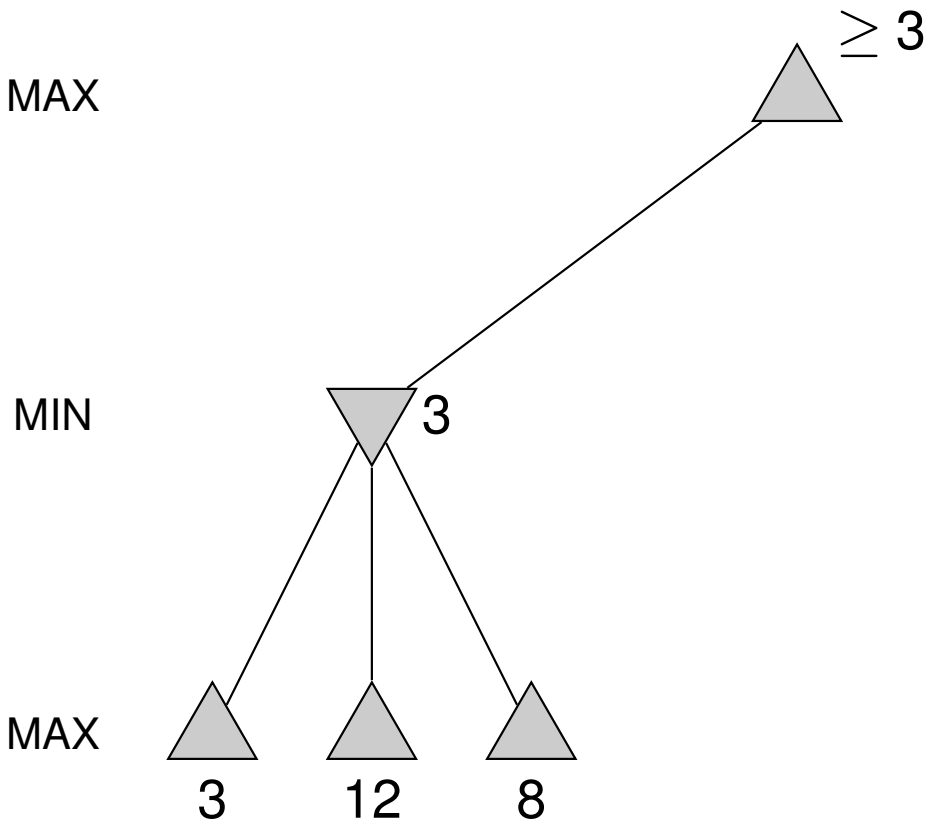
---

function MAX-VALUE(*state*, α, β) **returns** a *utility* value
  if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
  end if
  for each s in SUCCESSORS(*state*) **do**
    $\alpha \leftarrow max(\alpha, \text{MIN-VALUE}(s, \alpha, \beta))$
    if $\alpha \geq \beta$ **then return** β
    end if
  end for
end function

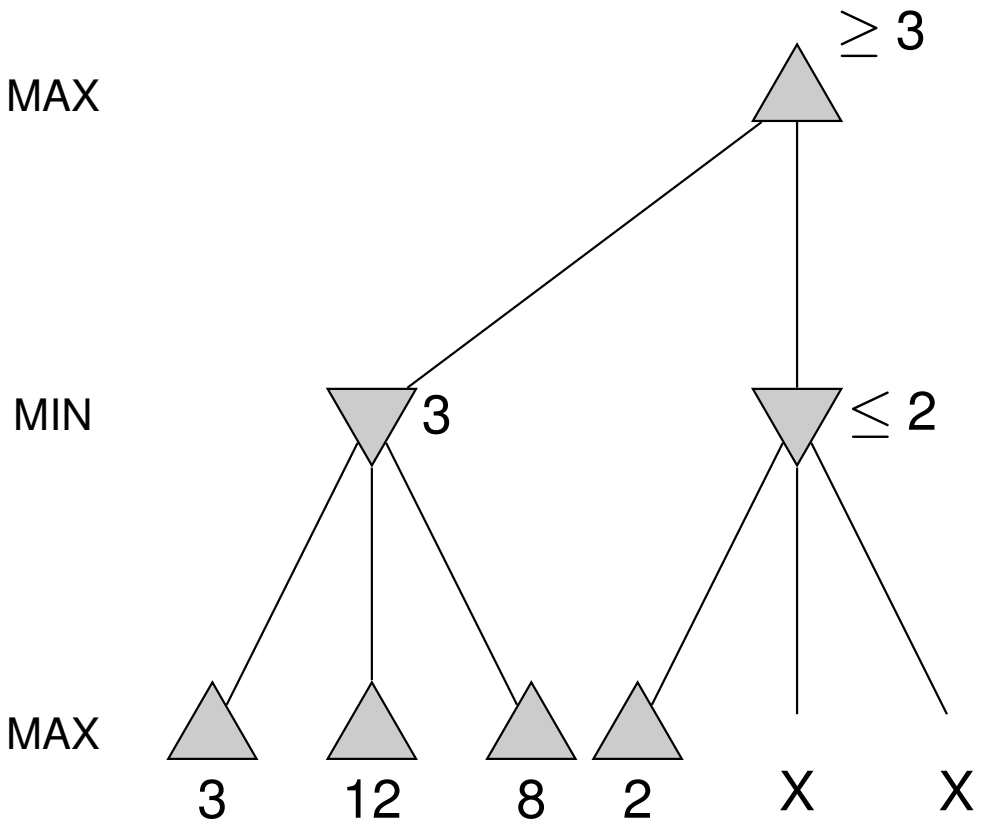
---

function MIN-VALUE(*state*, α, β) **returns** a *utility* value
  if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
  end if
  for each s in SUCCESSORS(*state*) **do**
    $\beta \leftarrow min(\beta, \text{MAX-VALUE}(s, \alpha, \beta))$
    if $\beta \leq \alpha$ **then return** α
    end if
  end for
end function

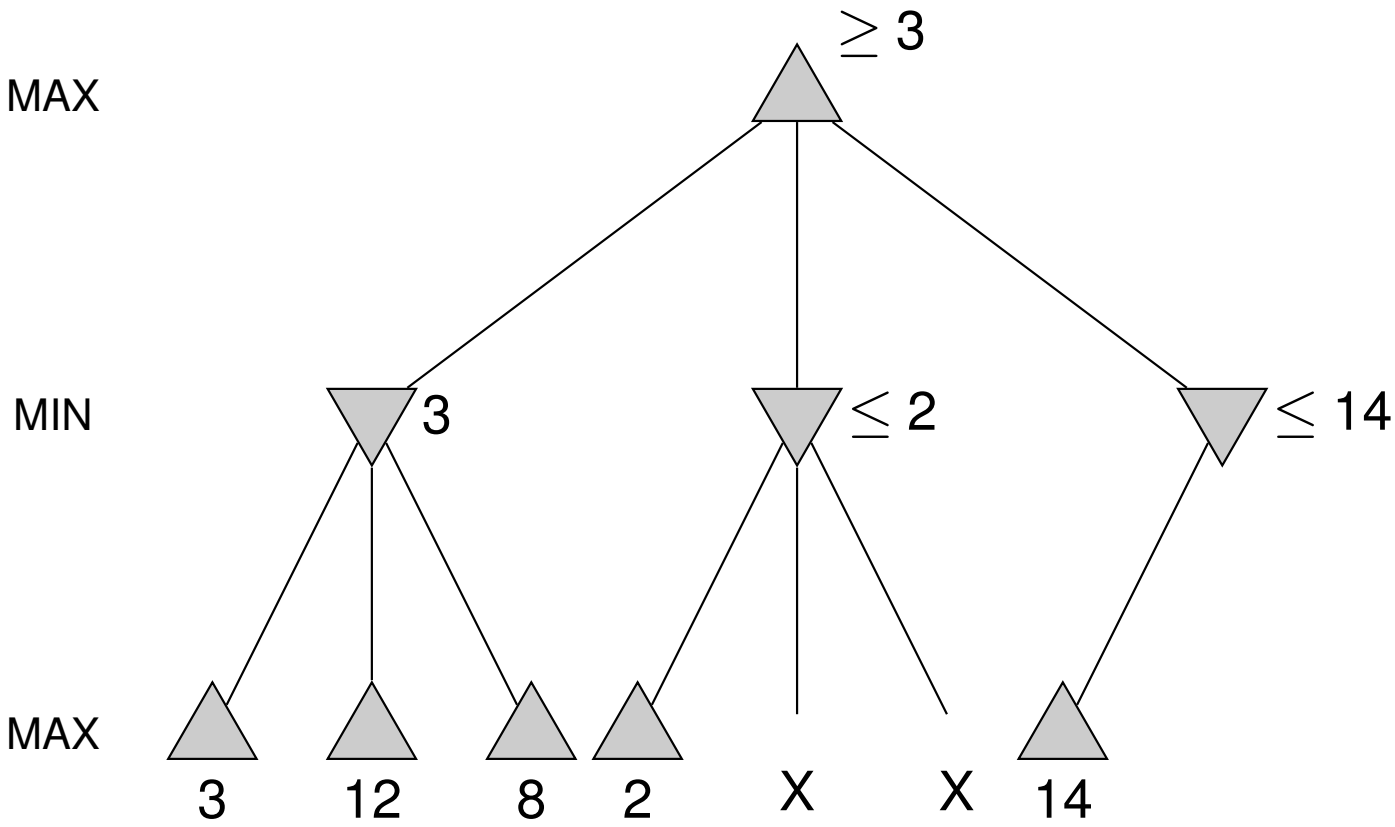
Strategia minimax z odcięciem α–β: przykład



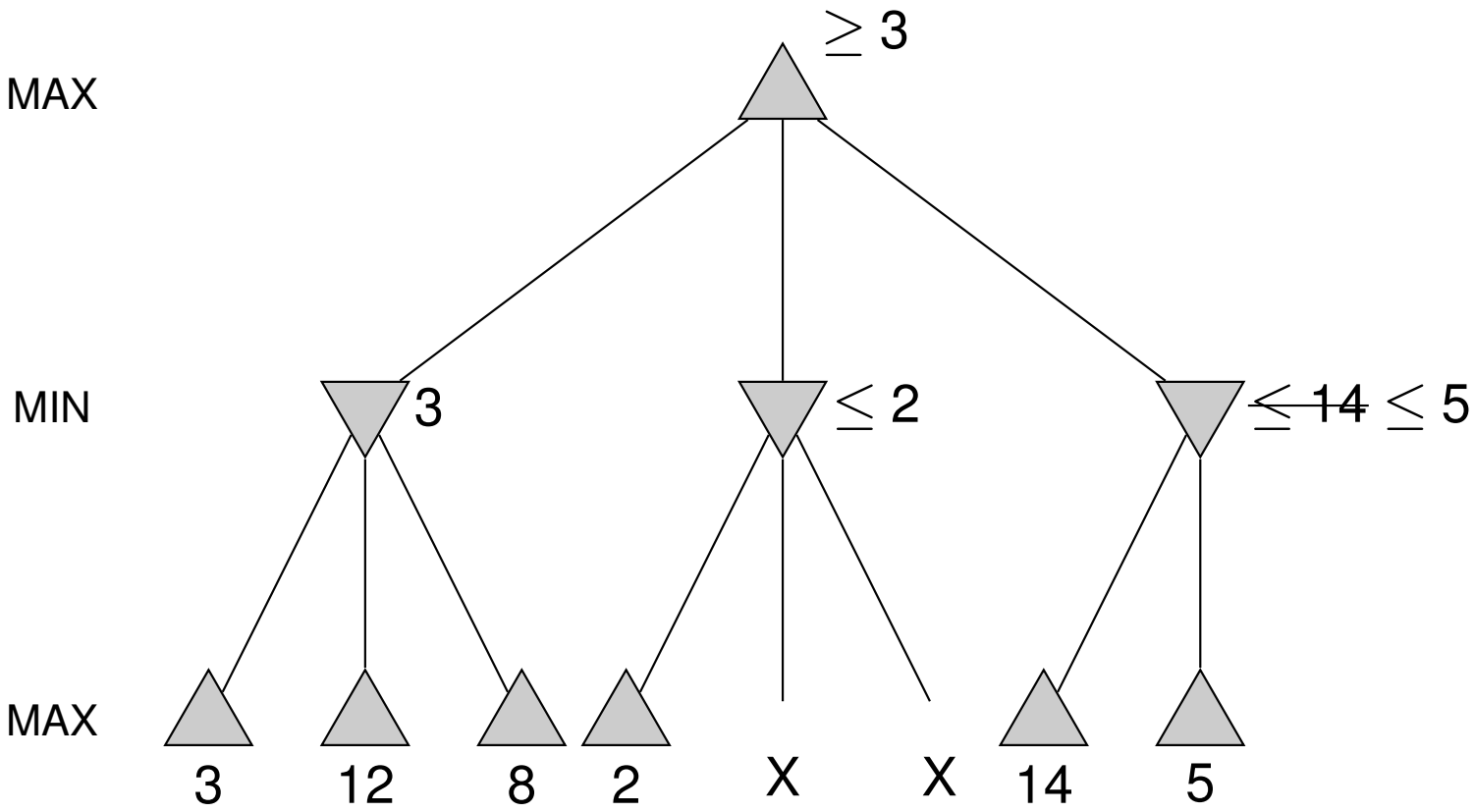
Strategia minimax z odcięciem α–β: przykład



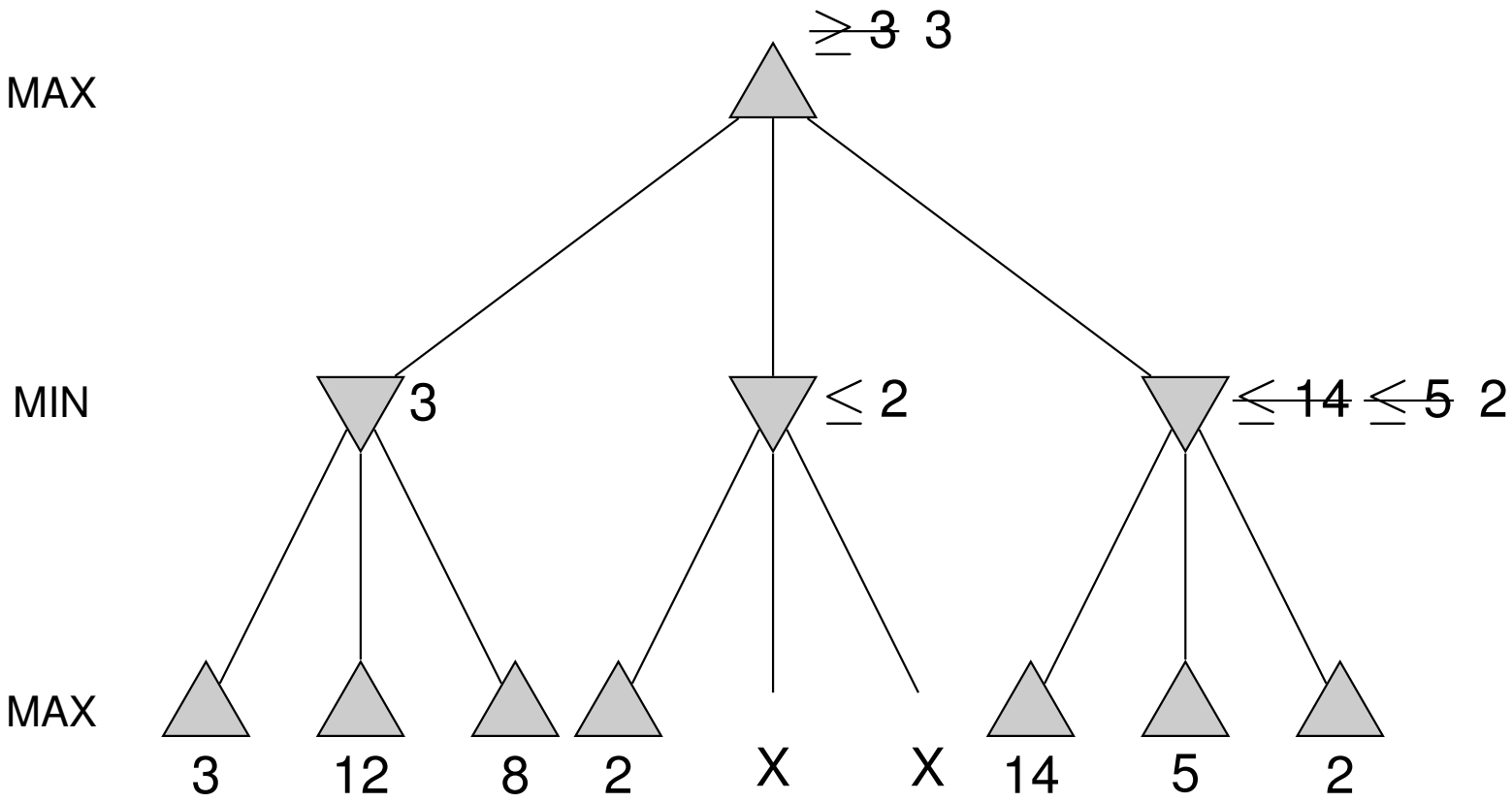
Strategia minimax z odcięciem α–β: przykład



Strategia minimax z odcięciem α–β: przykład



Strategia minimax z odcięciem α–β: przykład



Gry deterministyczne: osiągnięcia

Warcaby: Chinook zakończył 40-letnie panowanie mistrza świata Mariona Tinsley w 1994. Użył biblioteki wszystkich zakończeń dla 8 lub mniej pionków na planszy, w sumie 443.748.401.247 pozycji.
Chinook - warcaby zostały rozwiązane (Schaeffer et al, 2007).

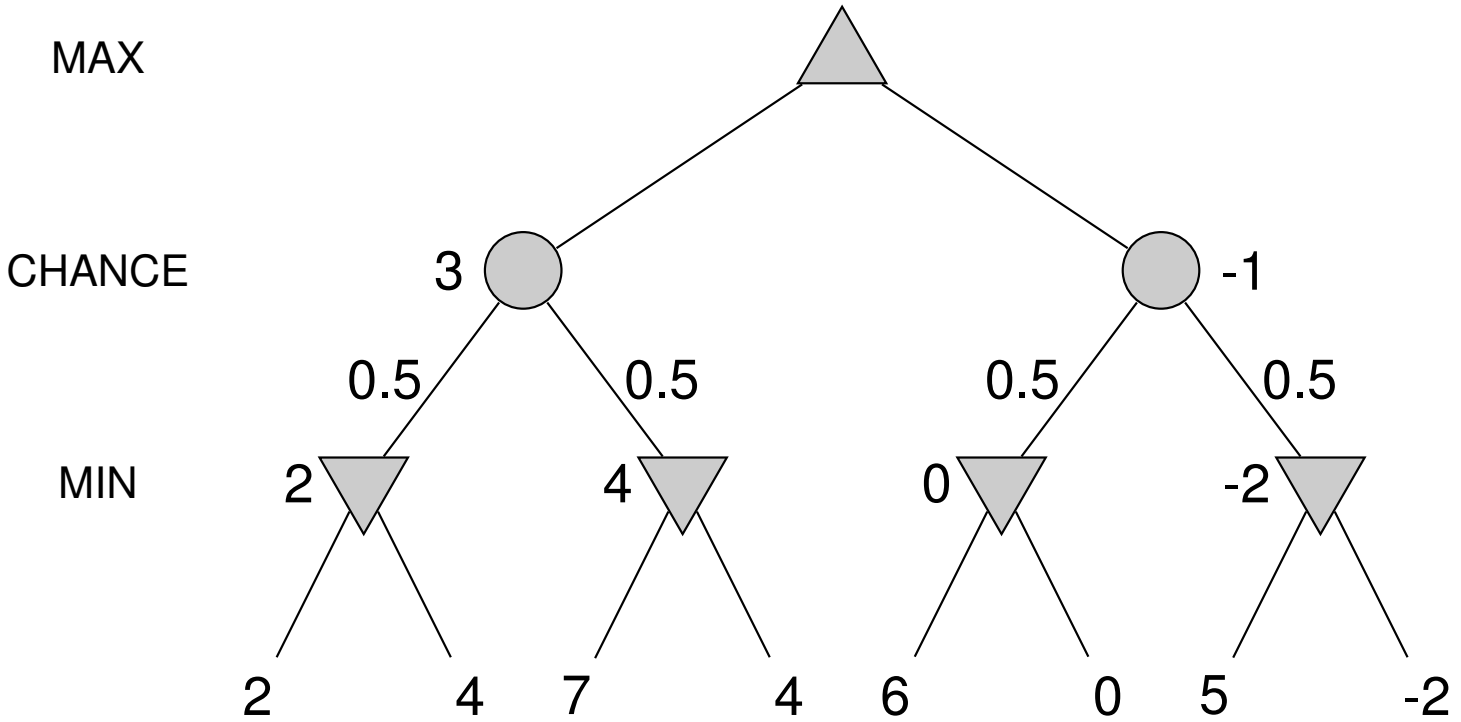
Szachy: Deep Blue pokonał mistrza świata Gary Kasparowa w meczu z 6-ioma partiami w 1997. Deep Blue przeszukiwał 200 milionów pozycji na sekundę, używając bardzo wyszukanej funkcji oceny i nieznanych metod rozszerzających niektóre ścieżki przeszukiwania do głębokości 40.

Go: mistrz świata odmówił rozgrywki z komputerami, które są zbyt słabe. W go, $b > 300$, więc większość programów używa bazy wiedzy z wzorcami do wyboru dopuszczalnych ruchów.

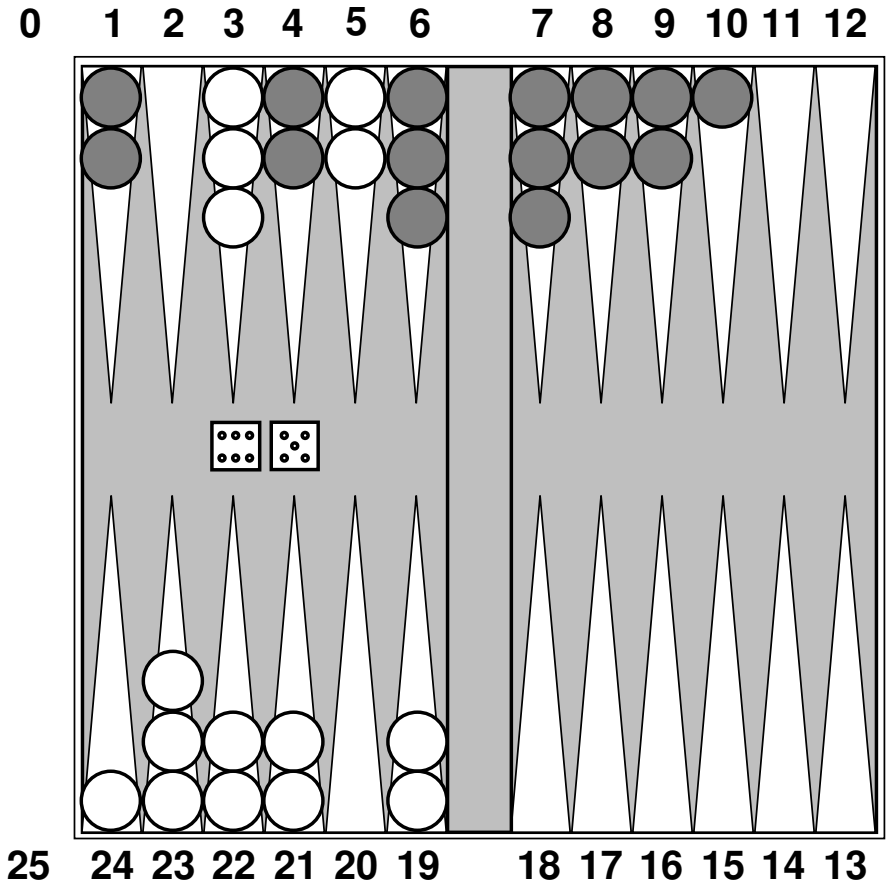
Gry niedeterministyczne

Źródło niedeterminizmu: rzut kostką, tasowanie kart

Przykład z rzucaniem monetą:



Gry niedeterministyczne: backgammon



Strategia uśrednionego minimax

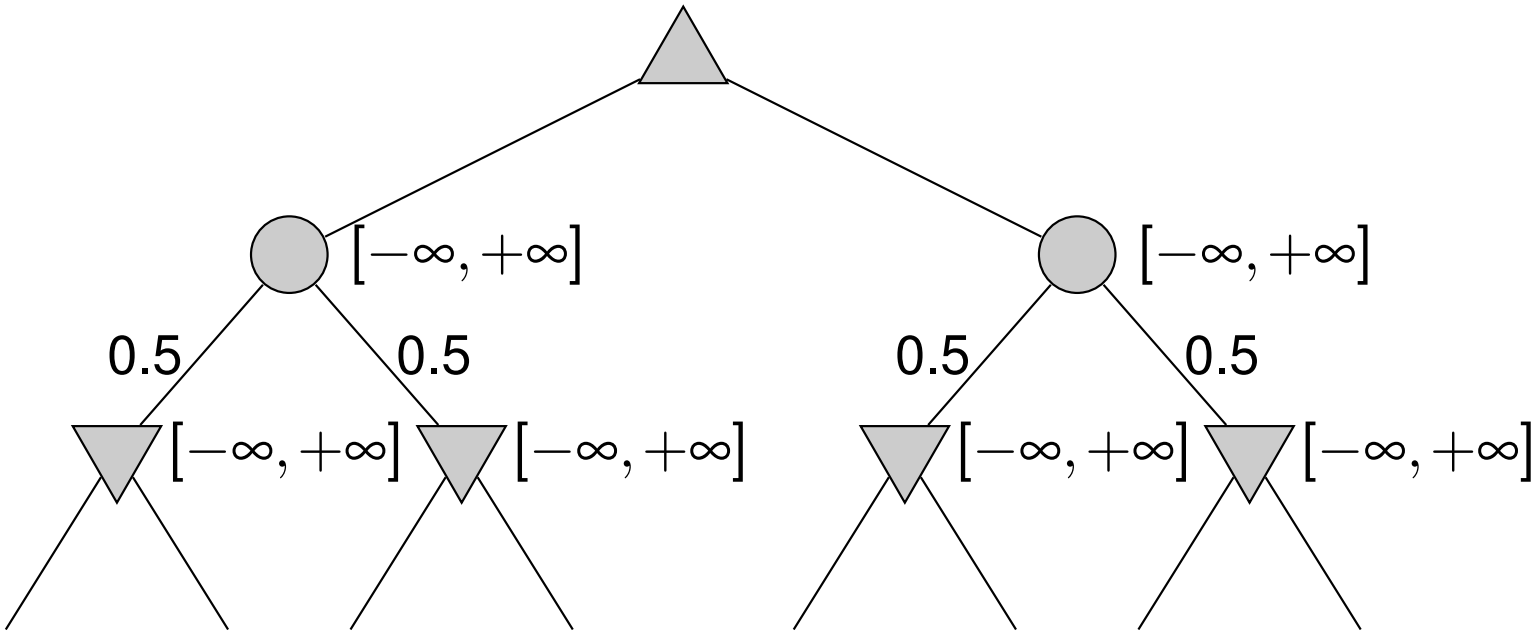
Uogólnienie strategii minimax dla gier niedeterministycznych

function EXPECTIMINIMAX-DECISION(*state*) **returns** an *action*
  action, *state* ← the *a*, *s* in SUCCESSORS(*state*) such that EXPECTIMINIMAX-VALUE(*s*) is maximized
  return *action*
end function

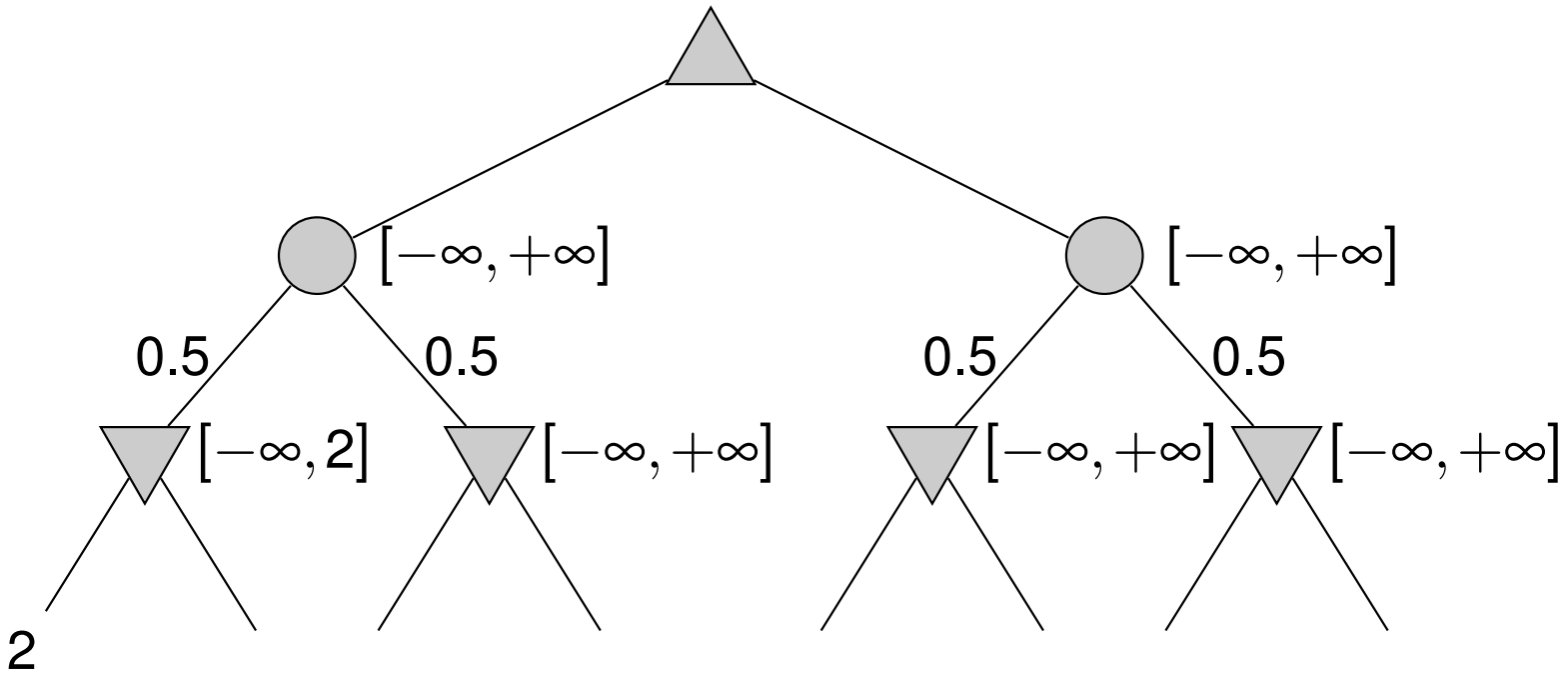
---

function EXPECTIMINIMAX-VALUE(*state*) **returns** a *utility* value
  if TERMINAL-TEST(*state*) **then**
    return UTILITY(*state*)
  else if *state* is a MAX node **then**
    return the highest EXPECTIMINIMAX-VALUE of SUCCESSORS(*state*)
  else if *state* is a MIN node **then**
    return the lowest EXPECTIMINIMAX-VALUE of SUCCESSORS(*state*)
  else if *state* is a chance node **then**
    return average of EXPECTIMINIMAX-VALUE of SUCCESSORS(*state*)
  end if
end function

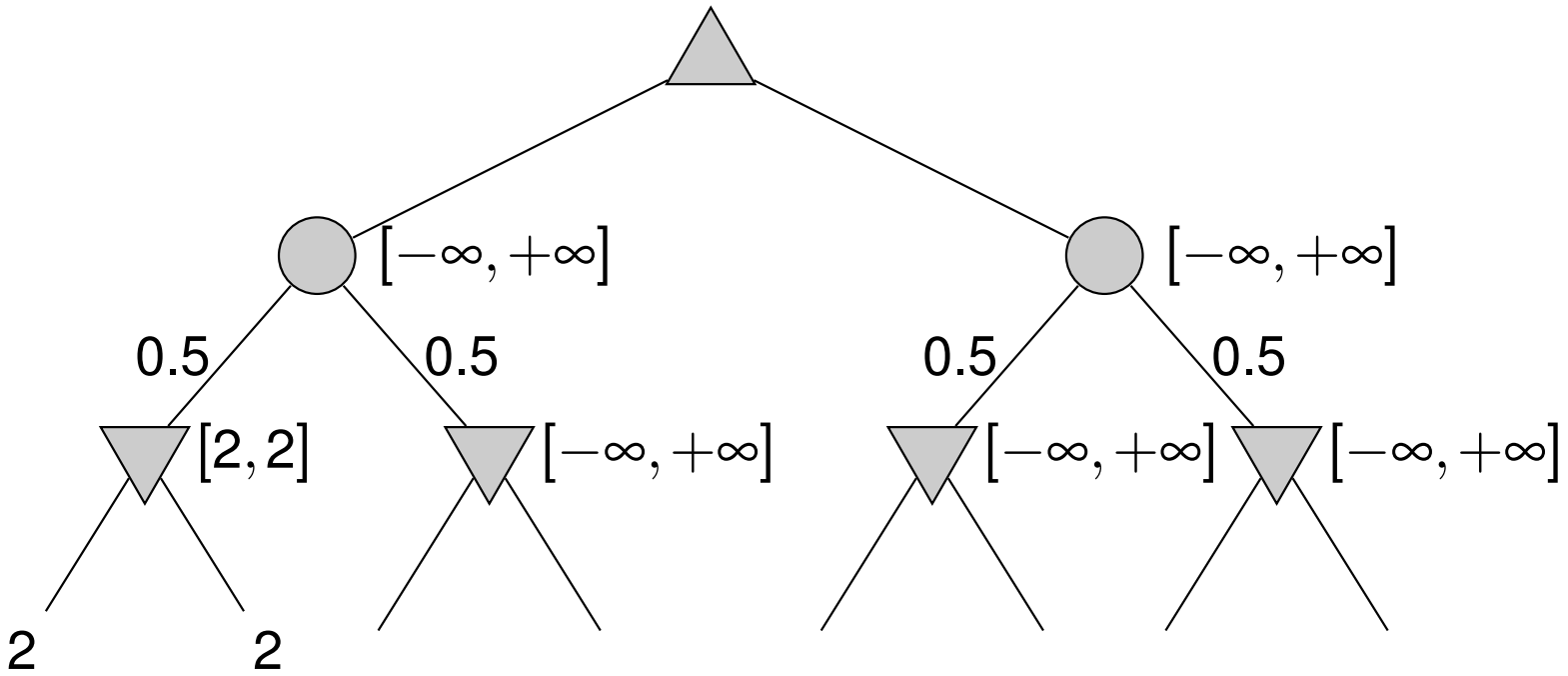
Strategia uśrednionego minimax z odcięciem α-β



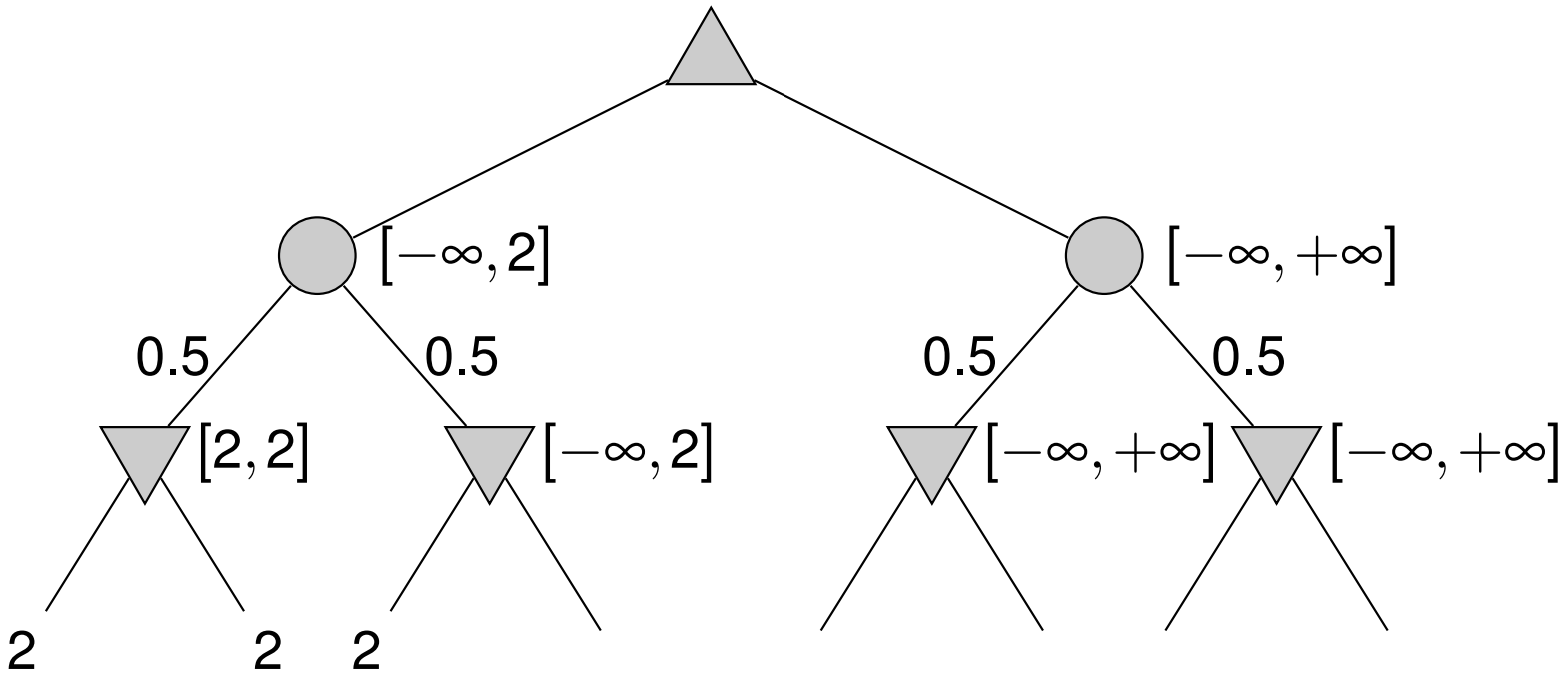
Strategia uśrednionego minimax z odcięciem α-β



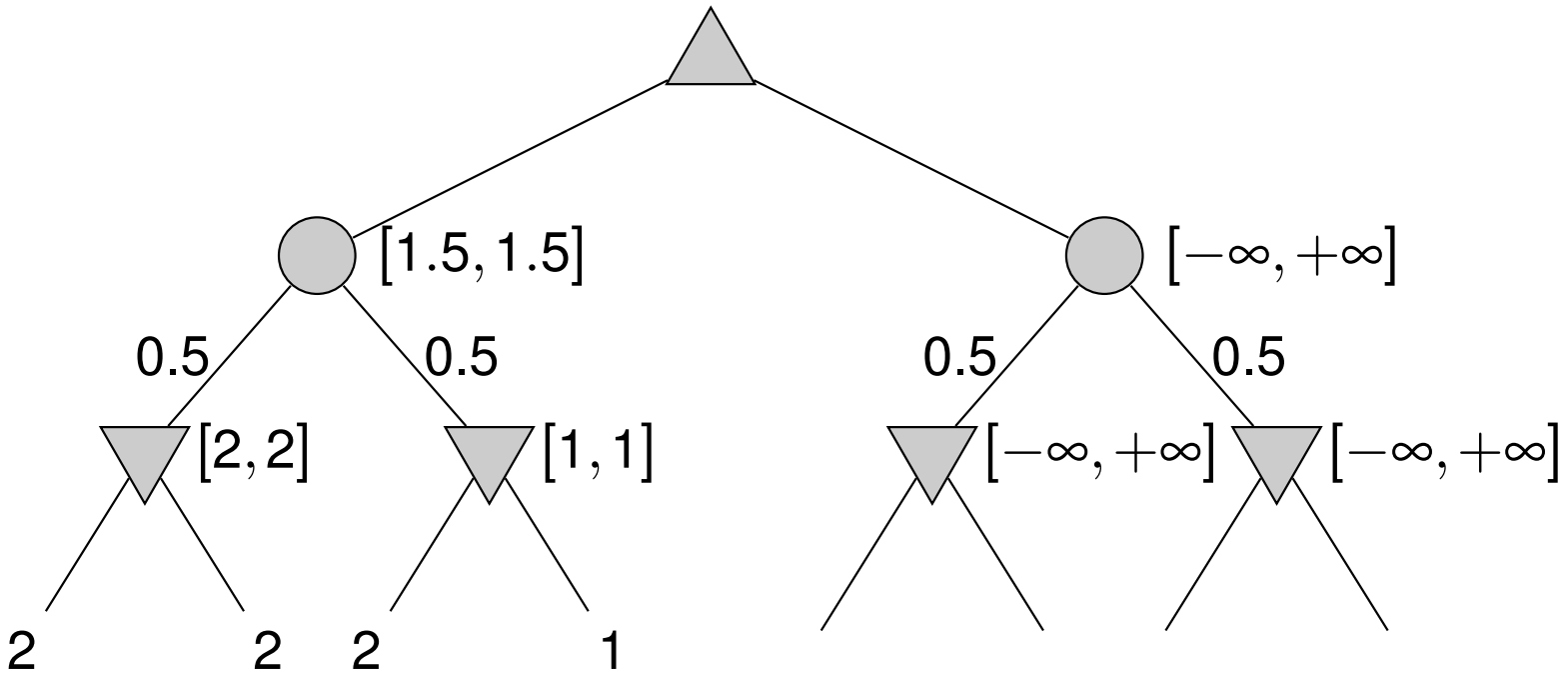
Strategia uśrednionego minimax z odcięciem α-β



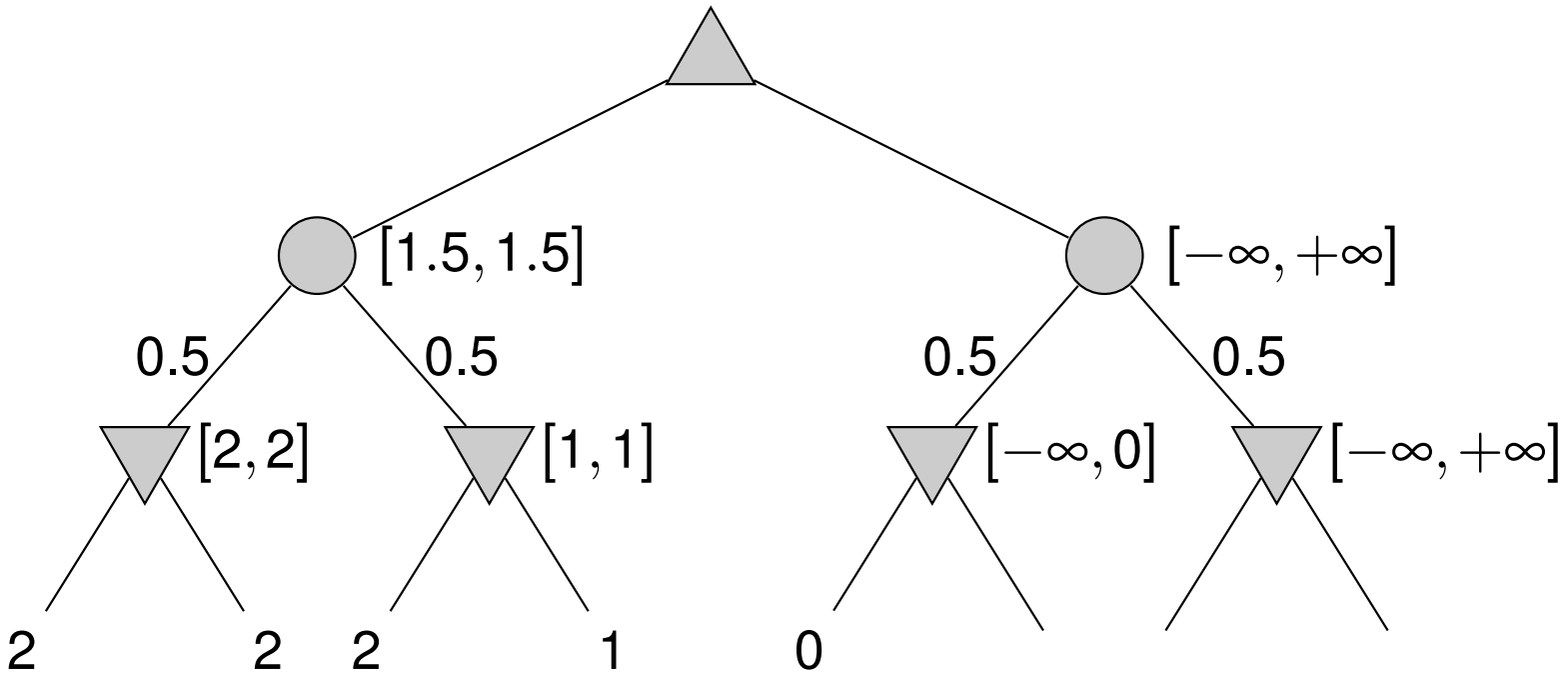
Strategia uśrednionego minimax z odcięciem α-β



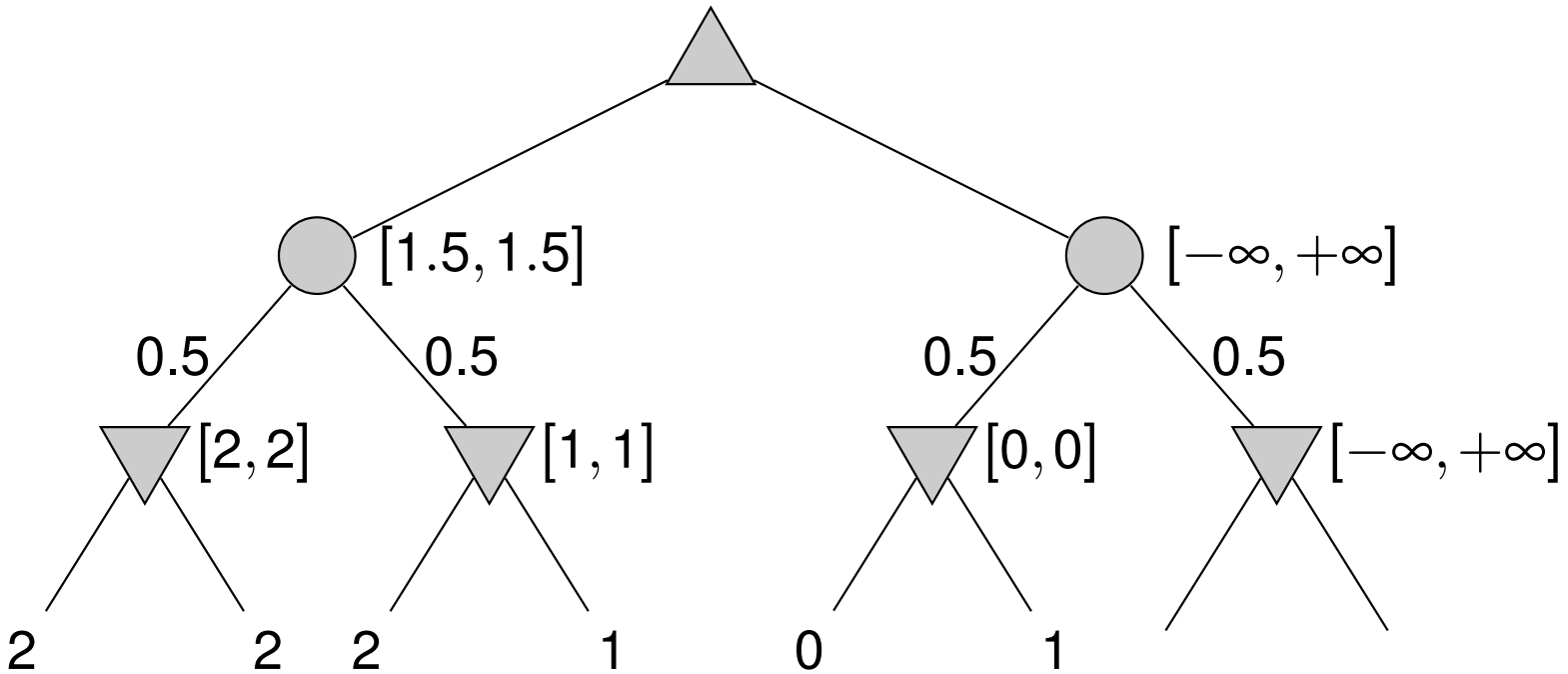
Strategia uśrednionego minimax z odcięciem α-β



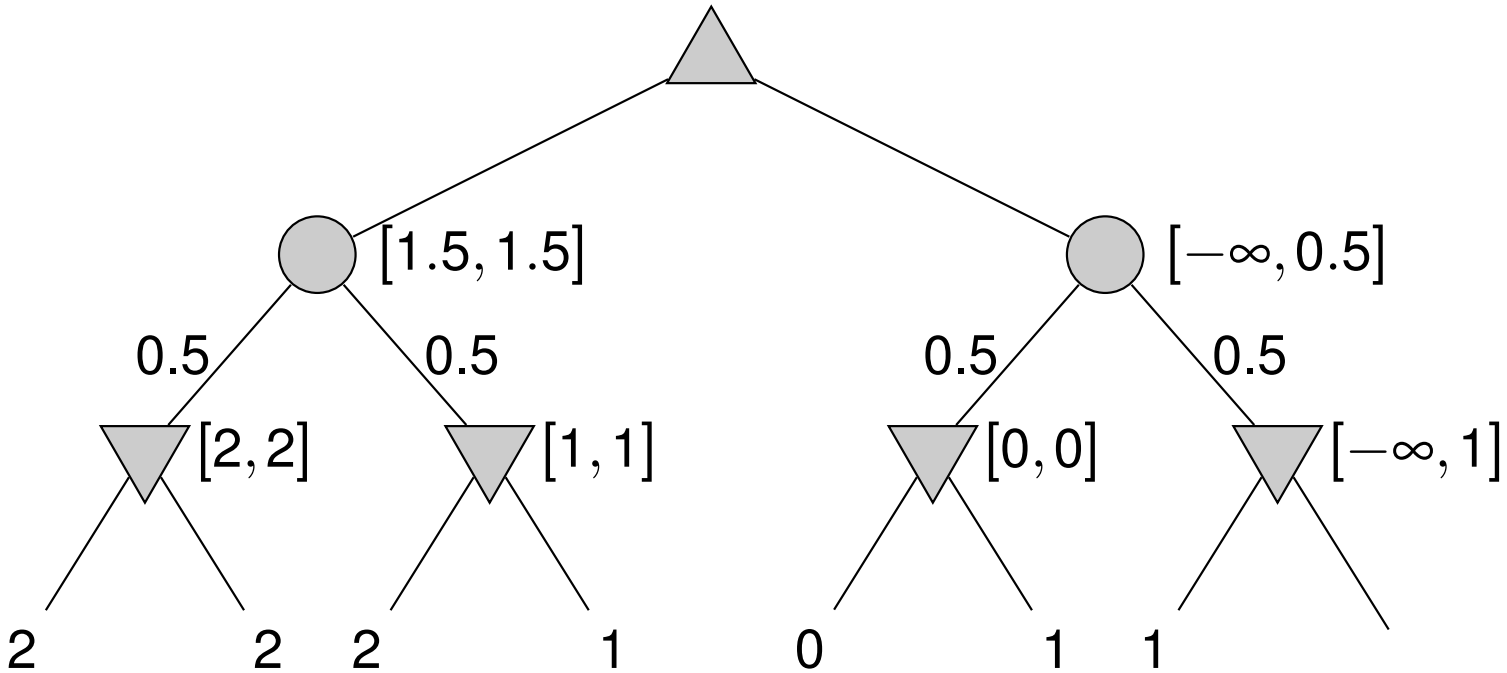
Strategia uśrednionego minimax z odcięciem α-β



Strategia uśrednionego minimax z odcięciem α-β

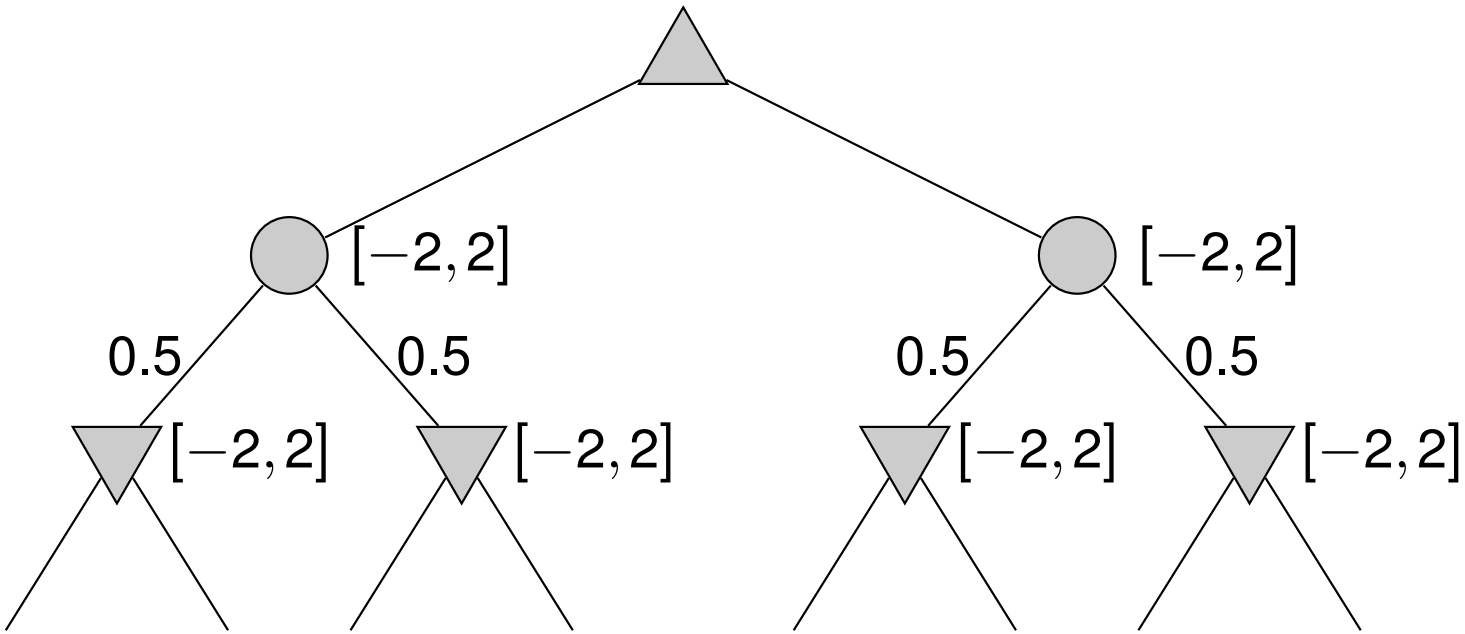


Strategia uśrednionego minimax z odcięciem α-β



Strategia uśrednionego minimax z odcięciem α-β

Bardziej efektywna, jeśli wartości wypłaty są ograniczone.



Gry z niepełną informacją

Np. gry karciane, w których początkowy zestaw kart przeciwnika jest nieznany.

Można policzyć prawdopodobieństwo każdego rozdania $\Rightarrow$ wygląda jak jeden "duży" rzut kostką na początku gry.

Pomysł: algorytm oblicza wartość minimax dla każdej akcji w każdym możliwym rozdaniu i wybiera akcje z największą wartością uśrednioną po wszystkich rozdaniach.

GIB, najlepszy program do brydża, przybliża tę ideę:

- generuje 100 rozdań zgodnych z informacją z licytacji
- wybiera akcję, która zbiera średnio najwięcej lew

Wyzwania w modelowaniu gier

- właściwy dobór strategii, ocena strategii
- adaptacja do informacji zyskiwanych podczas gry
- gry zespołowe
- gry z wieloma graczami
- modelowanie stylu gry przeciwników - mecze złożone z wielu partii/gier z tym samym przeciwnikiem, np. poker

Dziękuję za uwagę!