Gry

wykład 5

dr inż. Joanna Kołodziejczyk

jkolodziejczyk@wi.ps.pl

Zakład Sztucznej Inteligencji ISZiMM

Plan wykładu

Adversarial search — jak postępować, gdy inni agenci są naszymi przeciwnikami, czyli ich cele są przeciwstawne.

- Co to znaczy grać?
- Podejmowanie decyzji w grach dwuosobowych: min-max, $\alpha - \beta$
- Oszacowywanie pozycji
- Gry z elementami stochastycznymi
- Gry nieprecyzyjnej informacji

Co to są gry w kontekście SI?

Gry są formą środowiska wieloagentowego.

- Jaki jest cel i jak zachowują się inni agenci? Czy mają oni wpływa na nasze decyzje?
- Kooperacyjna i rywalizująca wersja środowiska wieloagentowego.

W grach mamy do czynienia z „nieprzewidywalnym” przeciwnikiem.
Typy gier (wg teorii gier):

- Gra dwuosobowa (dwóch agentów)
- Gra antagonistyczna (sumy zerowej) (wartości funkcji użyteczności agentów na końcu rozgrywki są równe lub przeciwne)
- Gra o skończonej liczbie strategii (środowisko deterministyczne)
- Gra niekoalicyjna
- Gra pełnej informacji (środowisko w pełni obserwowalne)

Kilka uwag

Gry były jednym z pierwszych zadań rozwiązywanych przez teoretyków SI. Do prekursorów badań nad szachami należy zaliczyć:

- Konrad Zuse — twórca pierwszego komputera i języka programowania wysokiego poziomu.
- Claude Shannon — twórca teorii informacji
- Norbert Wiener — twórca współczesnej teorii sterowania
- Alan Turing

Gry mają zazwyczaj duży współczynnik b szachy $b = 35$, gdy liczba ruchów do końca rozgrywki to około 50 dla każdego gracza, stąd pełne drzewo gry zawiera 35^{100} węzłów.

Gry wymagają podejmowania decyzji, gdy obliczenie decyzji optymalnej jest niewykonalne. W grach nie można sobie pozwolić na rozwiązania połowiczne.

Gry kontra problemy przeszukiwania

Przeszukiwanie - bez przeciwnika

- Rozwiązanie: odnalezienia stanu docelowego.
- Heurystyki pozwalają na znalezienie suboptymalnego a czasem optymalnego rozwiązania.
- Funkcja oceny: oszacowanie kosztu przejścia od punktu początkowego do rozwiązania przez dany węzeł.
- Przykłady: planowanie drogi, harmonogramowanie.

Gry - z przeciwnikiem

- Rozwiązaniem jest strategia (określa ruch dla każdej możliwej odpowiedzi oponenta).
- Ograniczenia czasowe wymuszają przybliżone rozwiązanie. W niewielu prostych grach można podać rozwiązanie optymalne.
- Funkcja oceny: oszacowanie „jakości” pozycji w grze.
- Przykłady: szachy, warcaby, trik-trak.

Podział gier

	deterministic	chance
perfect information	chess, checkers, go, othello	backgammon monopoly
imperfect information	battleships, blind tictactoe	bridge, poker, scrabble nuclear war

Początkowa część wykładu poświęcona jest grom deterministycznym o pełnej informacji.

Założenia

Dwóch graczy: MAX i MIN

MAX wykonuje ruch jako pierwszy, a nasępnie wykonuje naprzemienne ruchy z przeciwnikiem (ruch jednego gracza, to półruch (ang. ply)) aż do zakończenia gry. Zwycięzca otrzymuje nagrodę a przegrany karę.

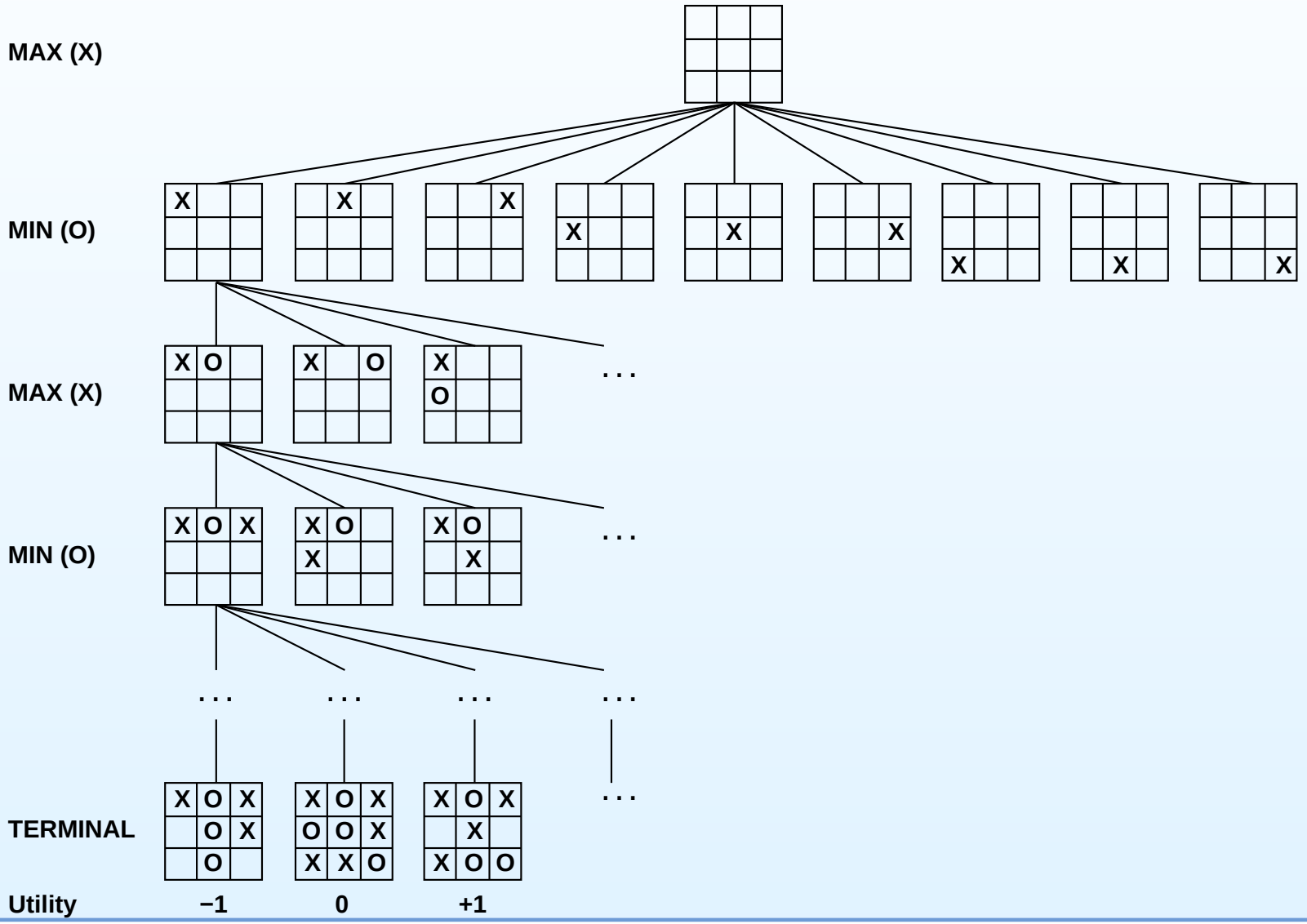
Gra jako problem przeszukiwania przestrzeni stanów:

- Stan początkowy: np. ustawienie początkowe na planszy i ustalenie, który gracz wykona pierwszy ruch.
- Funkcja generująca następny stan: tworzy listę par (ruch, stan) określającą dozwolone ruchy i otrzymane przez ten ruch stany.
- Warunek zakończenia: który określa kiedy gra jest zakończona. Stany w których gra się kończy nazywa się stanami terminalnymi.
- Funkcja użyteczności (funkcja wypłaty, celu): podaje numeryczną wartość węzła terminalnego np. wygrana (+1), przegrana (-1), remis (0) (kółko krzyżyk).

Stan początkowy i legalne ruchu definiują drzewo gry.

MAX użyje drzewa i przeszuka je w celu określenia kolejnego ruchu.

Drzewo gry



Strategie optymalne

Strategia optymalna dla MAX zakłada nieomylność oponenta MIN . Oznacza to, że zakładamy, że przeciwnik jest rozsądny i nie będzie wykonywał ruchów dla siebie niekorzystnych. Na przykładzie zostanie zademonstrowane jak znaleźć strategię optymalną, choć dla gier trudniejszych niż kółko i krzyżyk nie uda się jej obliczyć.

Założenie: Obaj przeciwnicy grają optymalnie.

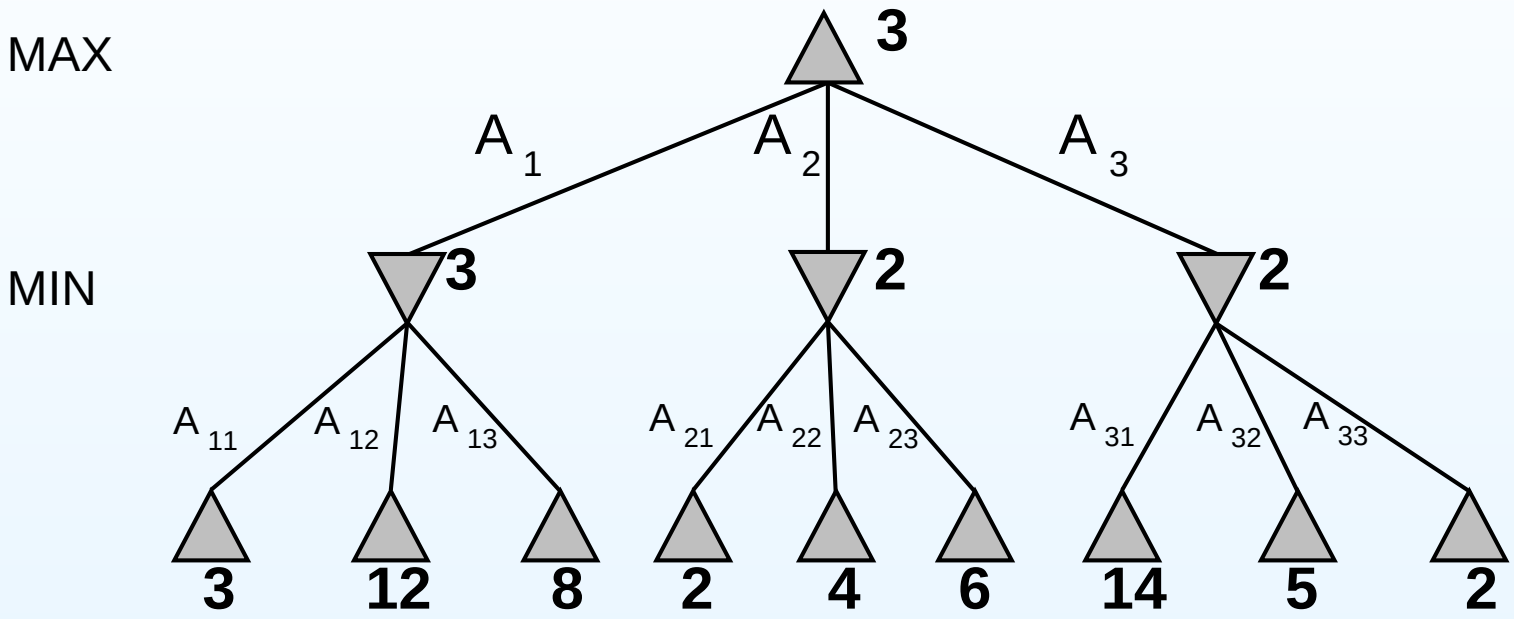
Dla danego drzewa gry, optymalna strategia zostanie określona przez sprawdzenie minimaksowej wartości (MINIMAX-VALUE(n)) każdego węzła.

Idea: wybierz ruch do pozycji o najkorzystniejszej wartości minimaksowej, która jest równa najlepszej osiągalnej funkcji wypłaty w grze z optymalnym przeciwnikiem. Stąd MAX będzie wybierał stany o wartościach maksymalnych a MIN o wartościach minimalnych.

MINIMAX-VALUE(n)=

1. UTILITY(n) If n is a terminal state
2. $max_{s \in successors(n)}$ MINIMAX-VALUE(s) If n is a max node
3. $min_{s \in successors(n)}$ MINIMAX-VALUE(s) If n is a min node

Przykład prostej gry dwuosobowej



W węzłach terminalnych podane są wartości wypłat dla graczy. Gracz MAX może wykonać ruchy A_1, A_2 lub A_3, gdy w odpowiedzi na ruch A_1 gracz MIN może wykonać ruchy A_{11}, A_{12} lub A_{13}, itd.

Decyzją minimax MAX wybierze ruch $A1$ ponieważ daje mu on najlepsze zyski przy optymalnej grze przeciwnika.

Co się stanie jeżeli...

Co się stanie jeżeli MIN nie zagra optymalnie?

Definicja o optymalnej (rozsądnej) grze gracza MAX zakłada również optymalność gry MIN co oznacza maksymalizację najgorszego możliwego wyniku dla MAX w węzłach MIN .

Zmiana strategii MIN na nieoptymalną nie pogorszy jednak wyników MAX . Może je tylko polepszyć. Dalczego?

Zapewne istnieją lepsze strategie suboptymnalne na przeciwnika suboptymalnego, ale będą sobie radziły gorzej z przeciwnikiem optymalnym.

Algorytm min-max

Algorytm min-max oblicz minimaksową decyzję dla bieżącego węzła.

Wykorzystuje rekurencyjne obliczenia wartości minimaksowej dla każdego węzła potomnego.

Rekurencja przebiega aż do momentu osiągnięcia liści w drzewie gry i przekazania tych wartości wstecz przez zwinięcie rekurencji.

Algorytm min-max

```
1  function MINIMAX-DECISION(state) returns an action
2    inputs: state, current state in game
3      v = MAX-VALUE(state)
4      return the action in SUCCESSORS(state) with value v
```

```
1  function MAX-VALUE(state) returns a utility value
2    if TERMINAL-TEST(state) then return UTILITY(state)
3    v = - inf
4    for all s in SUCCESSORS(state) do
5       v = MAX(v,MIN-VALUE(s))
6    return v
```

```
1  function MIN-VALUE(state) returns a utility value
2    if TERMINAL-TEST(state) then return UTILITY(state)
3    v = inf
4    for all s in SUCCESSORS(state) do
5       v = MIN(v,MAX-VALUE(s))
6    return v
```

Właściwości algorytmu min-max

1. Zupełność (completeness): czy zawsze znajdzie rozwiązanie jeżeli ono istnieje?
   Tak, jeżeli drzewo gry jest skończone
2. Złożoność czasowa (time complexity):
3. Obszar zajmowanej pamięci (space complexity):
4. Optymalność (optimality):

Właściwości algorytmu min-max

1. Zupełność (completeness):
   Tak, jeżeli drzewo gry jest skończone
2. Złożoność czasowa (time complexity): liczba węzłów generowana/rozwijana.
   $O(b^m)$
3. Obszar zajmowanej pamięci (space complexity):
4. Optymalność (optimality):

Właściwości algorytmu min-max

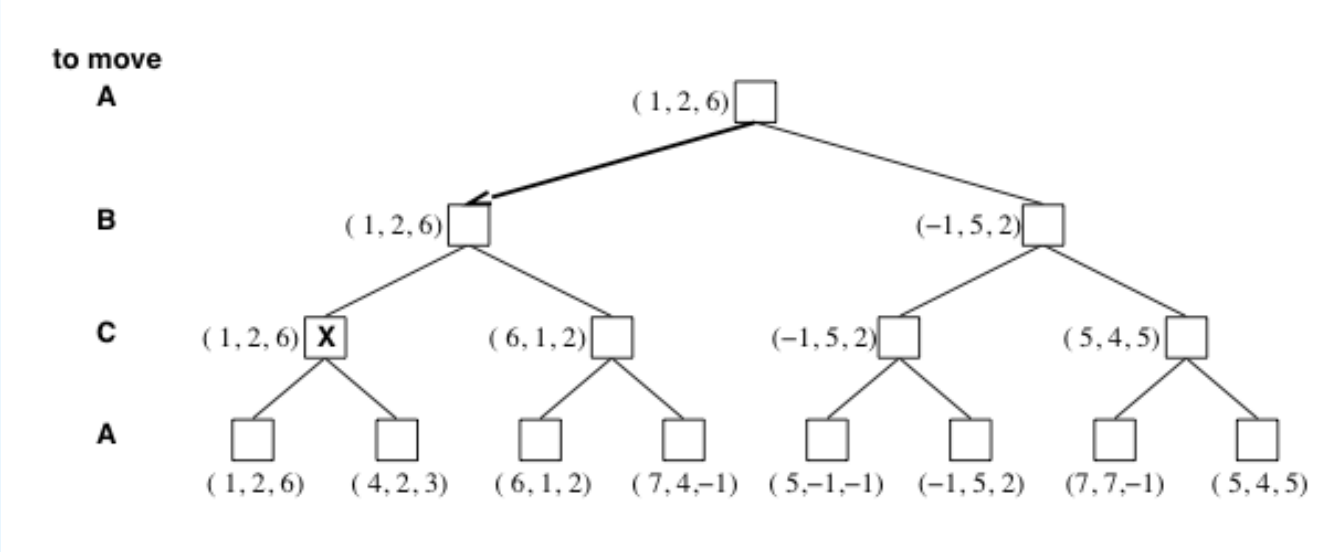
1. Zupełność (completeness):
   Tak, jeżeli drzewo gry jest skończone
2. Złożoność czasowa (time complexity):
   $O(b^m)$
3. Obszar zajmowanej pamięci (space complexity):
   maksymalna liczba węzłów w pamięci.
   $O(bm)$ Rozwinięcie DFS.
4. Optymalność (optimality):

Właściwości algorytmu min-max

1. Zupełność (completeness):
   Tak, jeżeli drzewo gry jest skończone
2. Złożoność czasowa (time complexity):
   $O(b^m)$
3. Obszar zajmowanej pamięci (space complexity):
   $O(bm)$ Rozwinięcie DFS.
4. Optymalność (optimality): czy zawsze znajdzie najmniej kosztowne rozwiązanie?
   Tak, jeżeli przeciwnik gra optymalnie.

Wieloosobowy min-max

Wiele popularnych gier zezwala na więcej niż dwóch graczy. Dla każdego węzła zamiast pojedynczej wartości pojawiają się wektory wartości, gdzie każda wartość oznacza wypłatę gracza np. dla graczy A, B, C, z każdym węzłem związany będzie wektor (v_A, v_B, v_C).



Na każdym poziomie gracz wybiera taki ruch, który prowadzi do jego największego zysku. Jeżeli pozostali gracze zagrają optymalnie, to otrzymane wskazanie będzie optymalne.

Wady algorytmu min-max

Liczba stanów gry rośnie wykładniczo do liczby ruchów.

Rozwiązanie: Nie rozwijaj wszystkich węzłów!

Przycinanie α–β

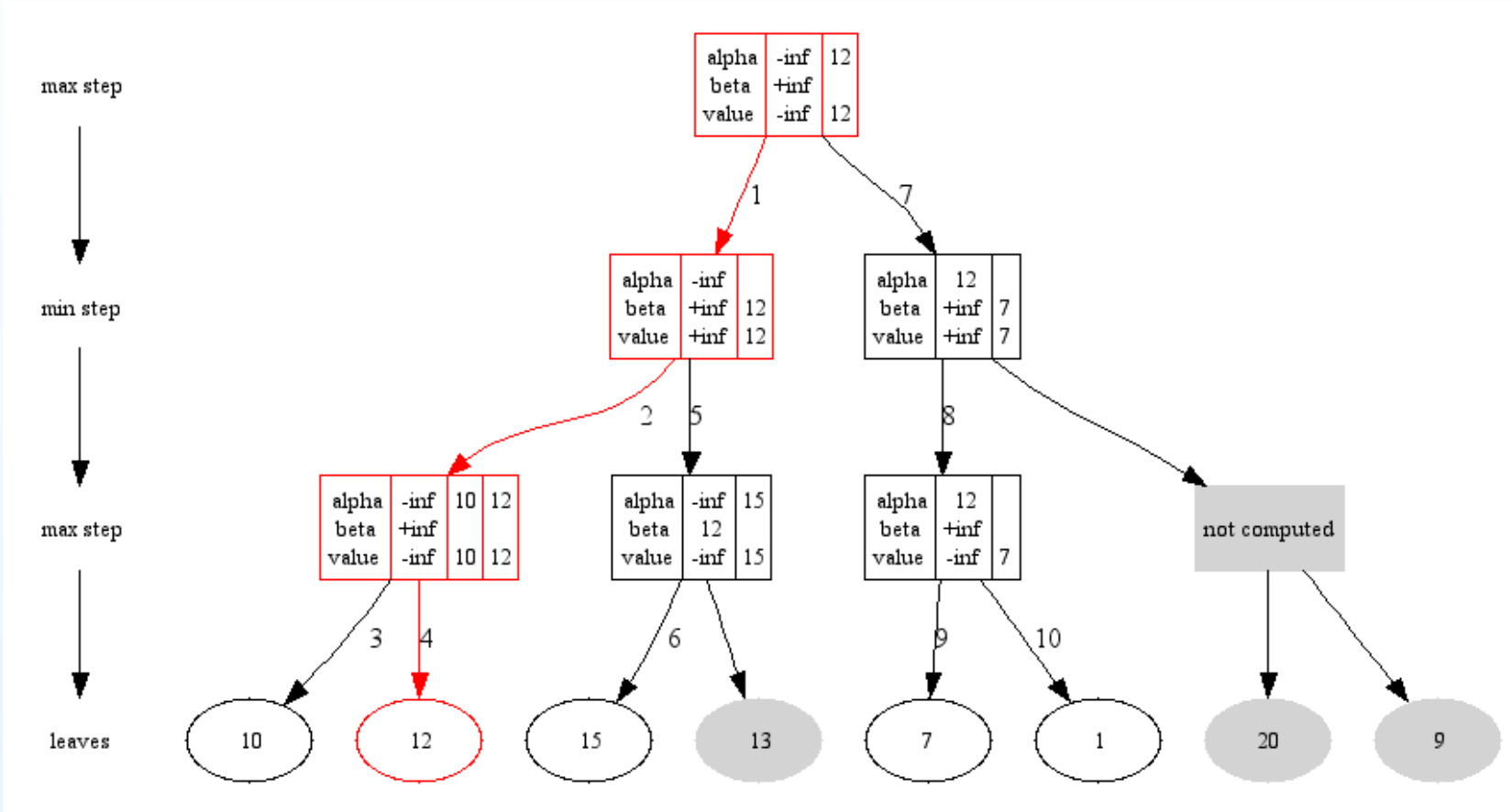
α = minimalny zysk, który jest pewny dla MAX (α zmienia się tylko na poziomie MAX). Wartość tylko rośnie.

β = maksymalny wynik dla MAX , który MAX ma nadzieję uzyskać grając przeciwko rozsądnemu przeciwnikowi. Wartość tylko maleje.

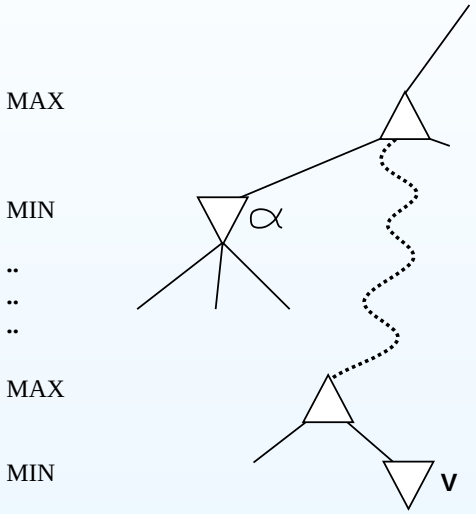
$\alpha \leq$ wypłata $\leq \beta$

Algorytm α–β cięć zawsze daje takie same rezultaty jak min-max!

Przykład przycinania α–β



Dlaczego przycinanie α–β ?



α najlepsza wartość (dla MAX) znaleziona na bieżącej ścieżce.

Jeżeli V jest gorsze od α, MAX będzie tego węzła unikał $\rightarrow$ przytnij gałąź.

Zdefiniuj β analogicznie dla MIN .

Algorytm α–β cięć (cz 1.)

```
1  function ALPHA-BETA-SEARCH(state) returns an action
2    inputs: state, current state in game
3      v = MAX-VALUE(state, -inf, +inf)
4      return the action in SUCCESSORS(state) with value v
```

```
1  function MAX-VALUE(state, alpha, beta) returns a utility value
2    inputs: state, current state in game
3          alpha, the best alternative for MAX along the path to state
4          beta, the best alternative for MIN along the path to state
5
6    if TERMINAL-TEST(state) then return UTILITY(state)
7    v = - inf
8    for all s in SUCCESSORS(state) do
9       v = MAX(v,MIN-VALUE(s, alpha, beta))
10      if v >= beta then return v
11      alpha = MAX(alpha, v)
12   return v
```

Algorytm α–β cięć (cz 2.)

```
1  function MIN-VALUE(state, alpha, beta) returns a utility value
2    inputs: state, current state in game
3          alpha, the best alternative for MAX along the path to state
4          beta, the best alternative for MIN along the path to state
5
6    if TERMINAL-TEST(state) then return UTILITY(state)
7    v = +inf
8    for all s in SUCCESSORS(state) do
9       v = MIN(v,MAX-VALUE(s, alpha, beta))
10      if v <= alpha then return v
11      beta = MIN(beta, v)
12   return v
```

Cechy algorytmy odcinania α–β

Odcinanie nie wpływa na wynik końcowy. Wskazany ruch jest taki sam jak przez agorytm min-max.

Efektywność odcina silnie zależy od kolejności w jakiej sprawdzani są potomkowie węzła bieżącego.

Gdyby zapewnić generowanie potomka o nalepszych cechach w pierwszej kolejności, to złożoność czasowa byłaby $O(b^{m/2})$. Oznacza to, iż α–β cięcięca pozwalają na zdwojenie głębokość przeszukiwania drzewa — algorytm może w tym samym czasie co min-max rozwinąć dwa razy więcej węzłów.

Losowe generowanie węzłów potomnych prowadzi do zmniejszenia złożoności czasowej do około $O(b^{3m/4})$.

Stany generowane w pewnej kolejności mogą się powtarzać. Aby uniknąć wielokrotnego rozwijania węzłów stosuje się tablice transpozycji przechowujące badane ścieżki w pamięci (CLOSED LIST w grafach).

Niestety dla szachów zastosowanie α–β cięć prowadzi zmniejszeni liczby węzłów 35^{50}, co nadal jest liczbą niemożliwą do rozwinięcia.

Co zrobić w przypadku dużego drzewa gry?

Skoro nie można rozwinąć drzewa do węzłów terminalnych, bo drzewo jest zbyt duże, to można:

- Rozwinąć drzewo do pewnej głębokości.
- Aby dowiedzieć się jaka jest użyteczność węzłów należy zastosować funkcję heurystyczną. Jej celem jest oszacowanie na ile korzystne będzie wykonanie ruchu.

Jeżeli program gra w szachy i zakładamy, że podany jest limit czasowy 100 sekund oraz że rozwija się 200^4 węzłów/sekundę

$\Rightarrow$ można przeszukać 200^6 węzłów na ruch co daje $\approx 35^{10/2}$
$\Rightarrow$ algorytm α–β cięć osiągnie $depth = 10$. Jest to poziom eksperta.

EVAL — Funkcja szacująca

Cel: oszacuj oczekiwane korzyści z gry od bieżącej pozycji.

Działanie algorytmu będzie silnie zależne od funkcji EVAL.

Wymagania:

- EVAL powinna ustalać taką samą kolejność węzłów końcowych jak sortowanie według wartości węzłów terminalnych.
- Obliczenia nie powinny być zbyt czasochłonne.
- Dla stanów nieterminalnych EVAL powinna być silnie skorelowana z wartością szansy na wygranie.

Z powodu odcięcia przeszukiwania do pewnego poziomu drzewa gry wprowadzamy niepewność oceny. Najczęściej wygląda to jak odgadywanie stanu końcowego z danej gałęzi.

Użyteczna tylko dla stanów „quiescent” (bez silnych wahań w wartości funkcji w najbliższym czasie).

Przykład funkcji szacującej: EVAL

W algorytmie α–β cięć należy zamienić linię 6 procedur MIN-VALUE i MAX-VALUE na (depth może być dobierane przez iteracyjne zagłębianie):

```
1  if CUTOFF-TEST(state, depth) then return EVAL(state)
```

W szachach EVAL może być ważoną sumą cech danej pozycji (funkcja liniowa)

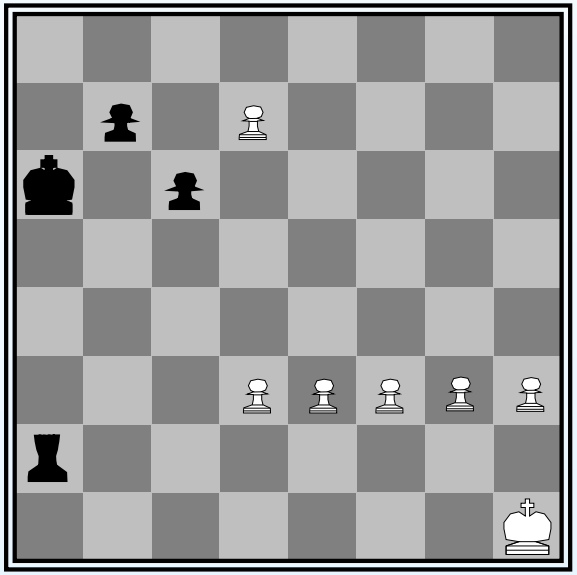
$$Eval(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$$

np., $w_1 = 9$ dla
$f_1(s)$ — [(liczba białych hetmanów) - (liczba czarnych hetmanów)].
$f_2(s)$ — ocena materialna: suma z wartości bierek na planszy (np. H=10, W =5, G=3.25, S=3.25, Pion=1).
$f_3(s)$ — struktura pionów (ocena strategiczna)
$f_4(s)$ — bezpieczeństwo króla (ocena strategiczna)
itd...

Dobór wag nie jest sprawą prostą i jest to zadnie optymalizacji. Stosowane są algorytmy uczenia się strategii gry, przez co łatwiejsze staje się wyznaczanie wartości w_i.

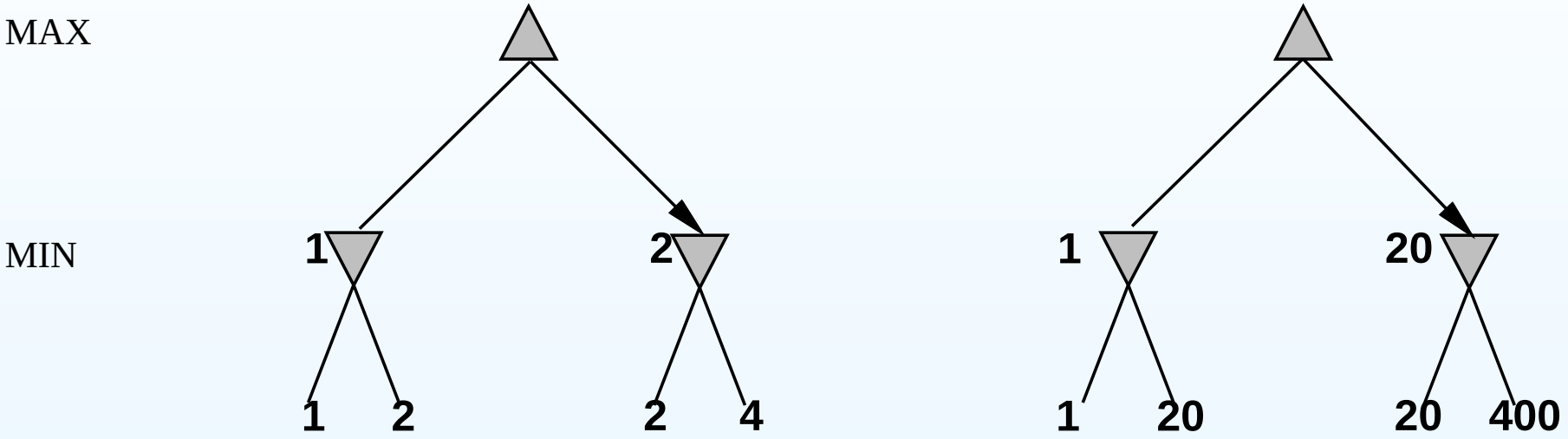
Efekt horyzontu

Ograniczona głębokość przeszukiwania może pominąć różne zagrożenia, które wykrytoby zwiększając głębokość przeszukiwania np. wynikające z linii przemiany.



Black to move

Wartość dokładna nie ma znaczenia



Wynik w grze zostanie zachowany przy każdym monotonicznym przekształceniu EVAL.

Istotna jest tylko kolejność.
Wypłata w grach deterministycznych zachowuje się jak funkcja porządkująca wg rosnących wartości.

Gry deterministyczne w praktyce

Warcaby: Chinook zakończył 40-letnie panowanie człowieka w tej grze (Marion Tinsley w 1994). Zastosowanie bazy końcówek opisujących idealną grą dla wszystkich pozycji zawierających 8 lub mniej pionków na szachownicy (całkowita liczba 443,748,401,247 pozycji).

Szachy: Deep Blue pokonał arcymistrza szachowego i mistrza świata Gary Kasparowa w 1997 (wynik 3,5:2,5). Deep Blue przeszukuje 200 milionów pozycji na sekundę, używa bardzo złożonej funkcji oszacowania i nieujawnioną metodę rozszerzania niektórych linii przeszukań do 40 półruchów.

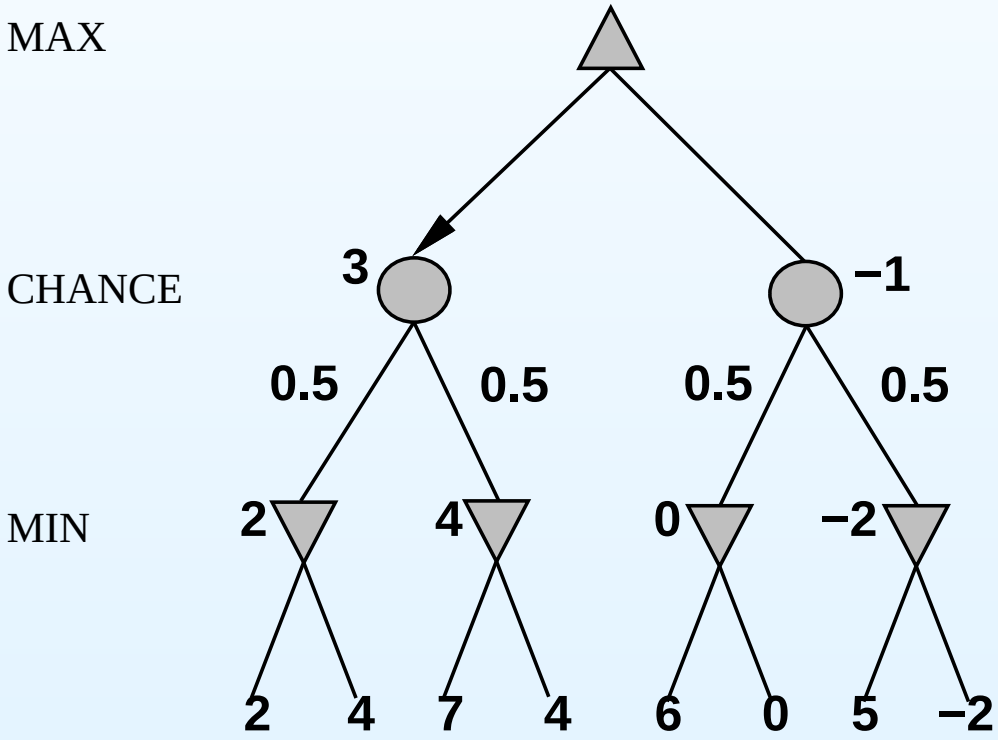
Reversi: mistrzowie odmówili rozgrywek z komputerami, które są zbyt dobre.

Go: mistrzowie odmówili rozgrywek z komputerami, które są zbyt słabe. W go $b > 300$ stąd większość programów używa baz wiedzy z wzorcami sugerującymi najbardziej obiecujące ruchy.

Podstawy gier niedeterministycznych

W grach niedeterministycznych przypadkowość wprowadza rzut kością, tasowanie kart.

Uproszczony przypadek rzutu monetą:



Expected minimax value

Jak podejmować poprawne decyzje gdy w grze występuje element losowości?

Expected mini-max value to uogólnienie wartości minimaksowej dla gier niedeterministycznych.

W węzłach deterministycznych MIN i MAX zasada postępowania jest taka sama natomiast w niedeterministycznych liczmy średnią ważoną ze wszystkich możliwych stanów zdarzenia losowego. Wagę stanowią prawdopodobieństwa.

EXPECTED-MINIMAX-VALUE(n)=

UTILITY(n) If n is a terminal state

$max_{s \in successors(n)}$ MINIMAX-VALUE(s) If n is a max node

$min_{s \in successors(n)}$ MINIMAX-VALUE(s) If n is a min node

$\sum_{s \in successors(n)} P(s) \cdot$ EXPECTEDMINIMAX(s) If n is a chance node

Gra niedeterministyczna w praktyce

Rzucanie kości zwiększa rozgałęzienia w drzewie b. Dla dwóch kości to 21 możliwych układów.

Trik-trak $\approx$ 20 dozwolonych posunięć

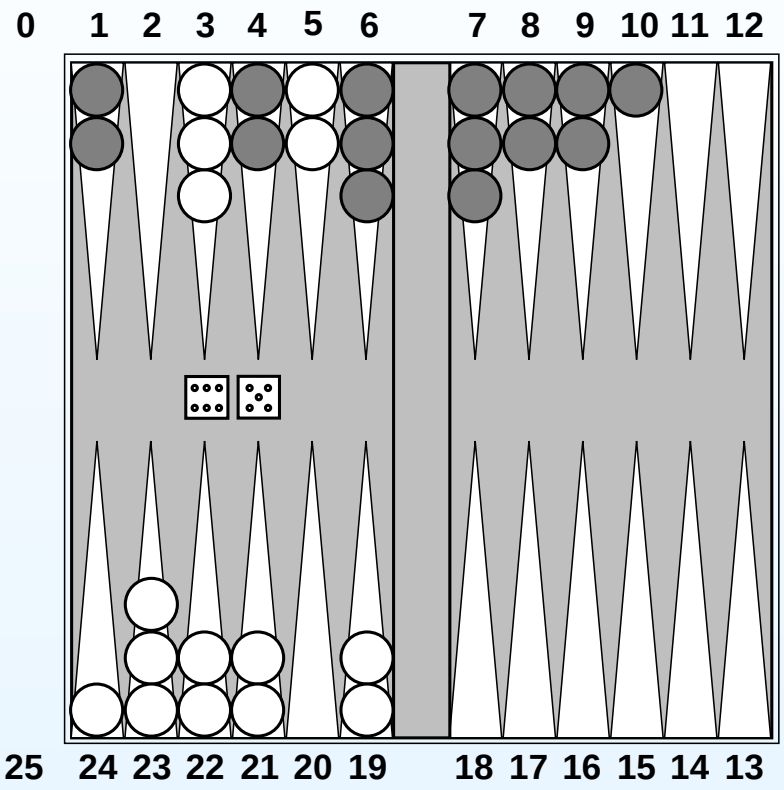
$$\text{depth } 4 = 20 \times (21 \times 20)^3 \approx 1.2 \times 10^9$$

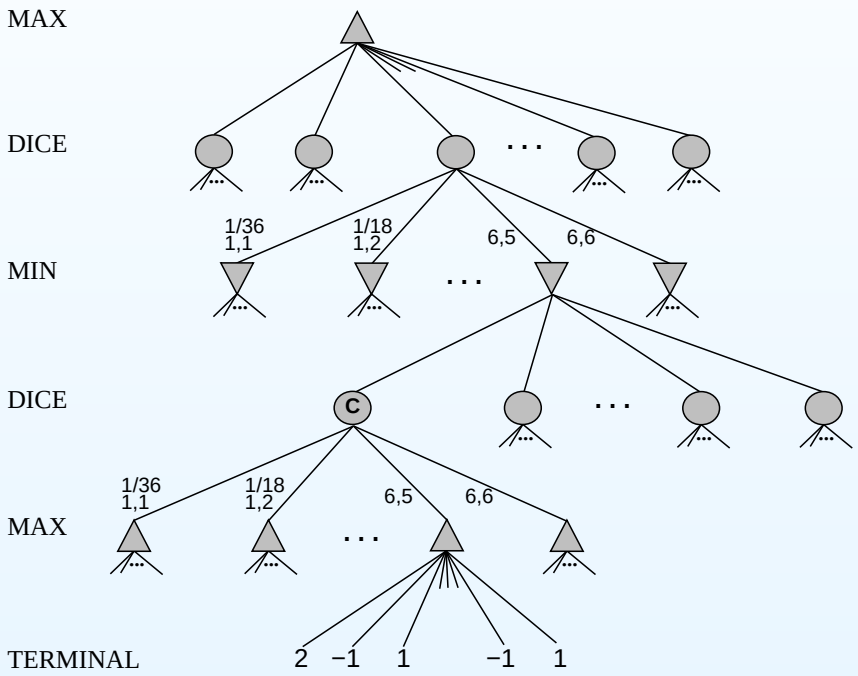
Kiedy głębokość wzrasta, prawdopodobieństwo osiągnięcia danego węzła zmniejsza się
$\Rightarrow$ wartość przewidywania się kurczy.

Cięcie α–β jest jeszcze bardziej nieefektywne.

TDGAMMON stosuje przeszukiwanie do poziomu 2 + bardzo dobra funkcja oceny $\approx$ poziom mistrza świata.

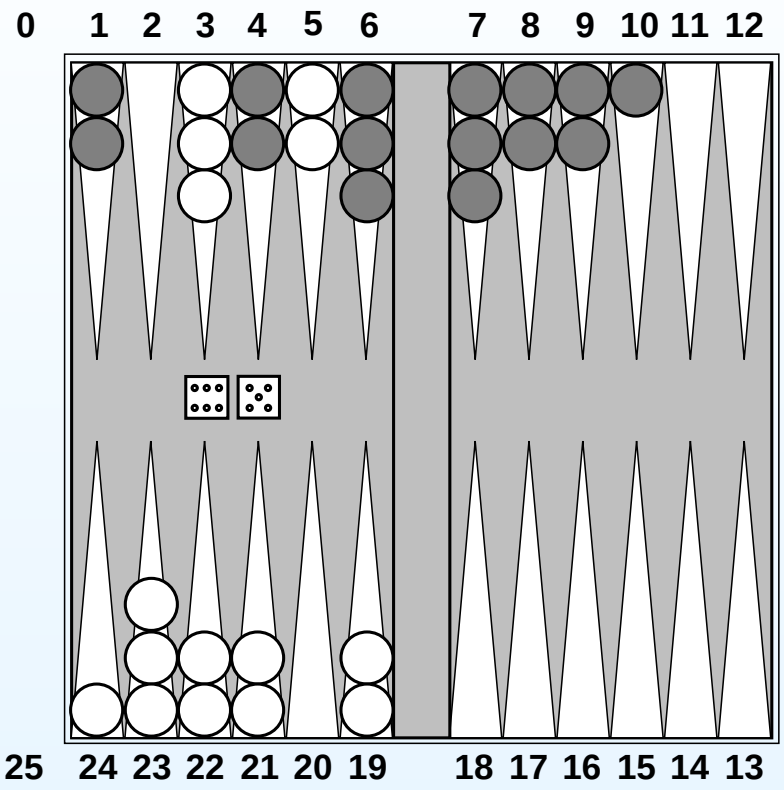
Przykład dla trik-traka

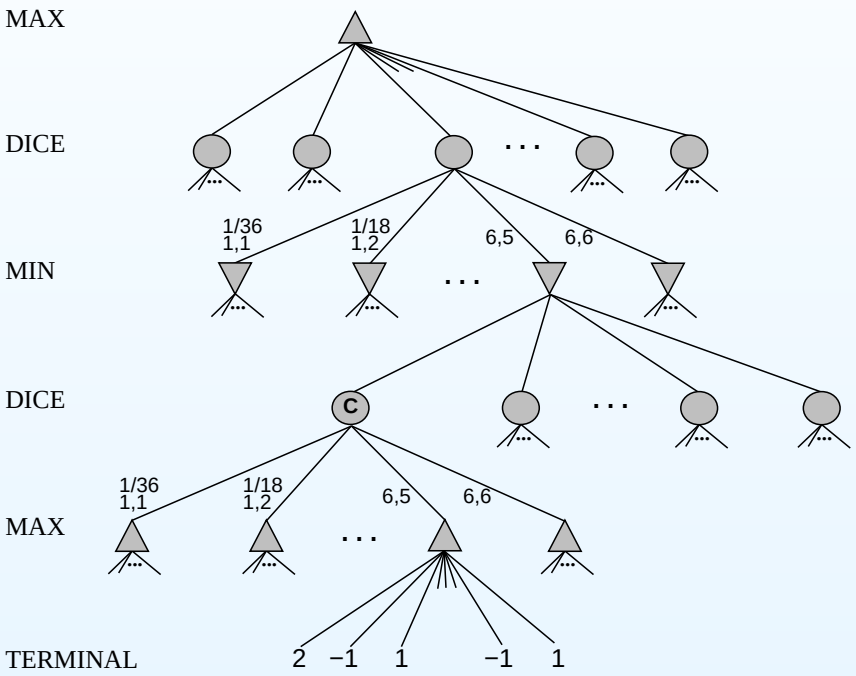




Możliwe posunięcia (5-10,5-11), (5-11,19-24),(5-10,10-16) i (5-11,11-16).

Przykład dla trik-traka

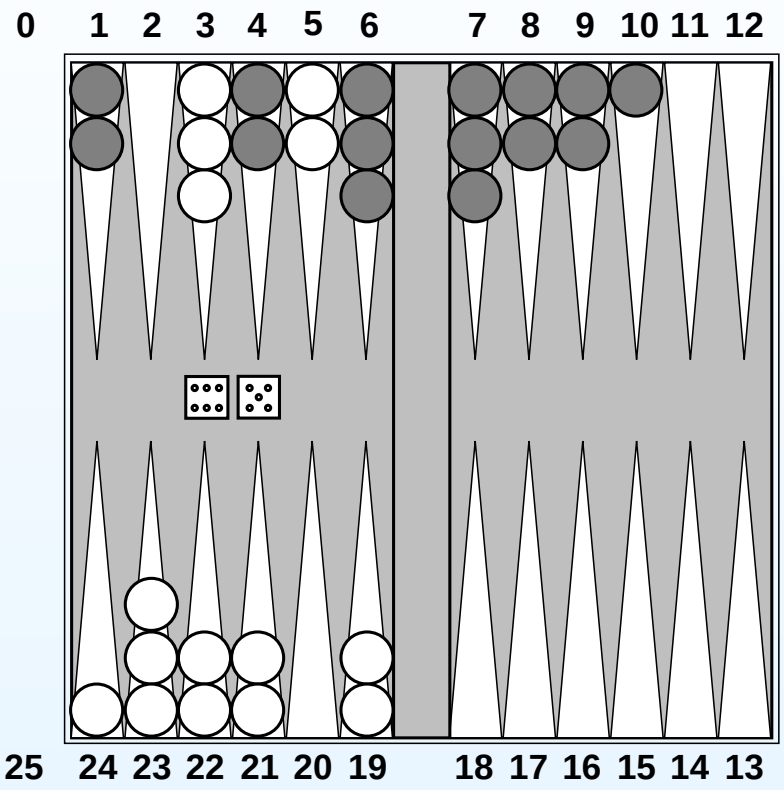


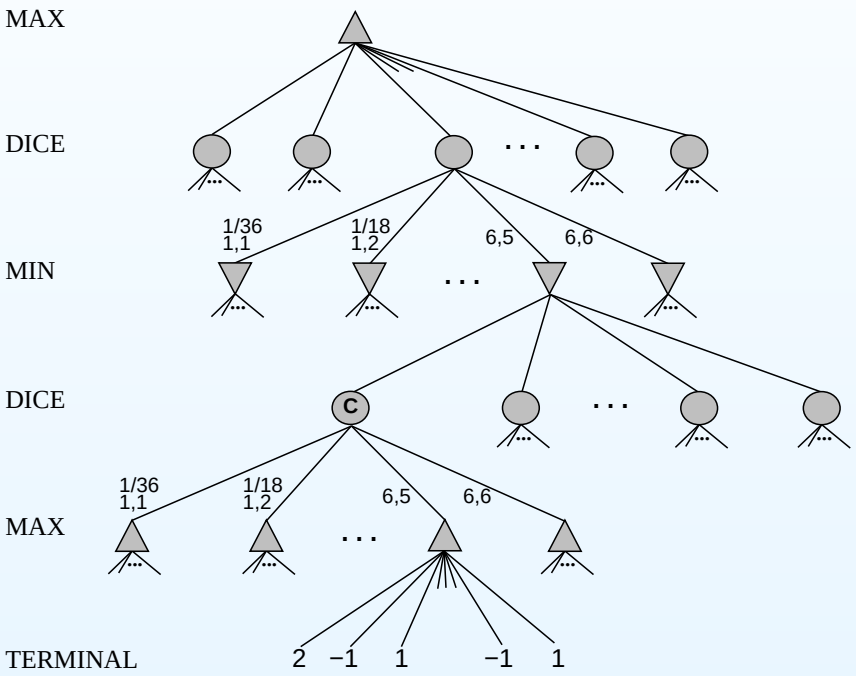


Możliwe posunięcia (5-10,5-11), (5-11,19-24),(5-10,10-16) i (5-11,11-16).

[1,1], ... ,[6,6] prawdopodobieństwo 1/36, pozostałe 1/18 .

Przykład dla trik-traka



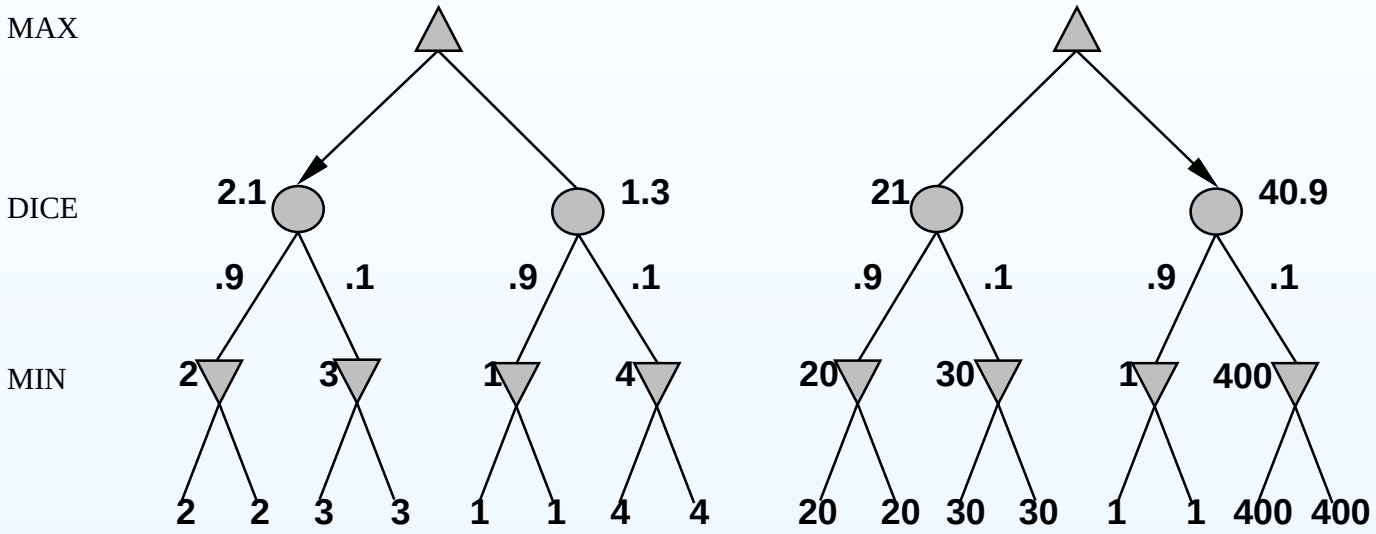


Możliwe posunięcia (5-10,5-11), (5-11,19-24),(5-10,10-16) i (5-11,11-16).

[1,1], ... ,[6,6] prawdopodobieństwo 1/36, pozostałe 1/18 .

Nie można obliczyć dokładnej wartości minimaxowej tylko wartość oczekiwaną.

Pozytywne oszacowanie



Lewy $\Rightarrow$ wykonaj ruch A1
Prawy $\Rightarrow$ wykonaj ruch A2

Jeżeli funkcja EVAL przeskaluje wartości wypłaty, to nie zmieni to decyzji o wykonaniu ruchu.

Poprawne zachowanie zapewni tylko liniowa transformacja EVAL.

Gry o niepełnej informacji

Przykładem może być gra w karty, gdy początkowy układ kart przeciwnika nie jest znany.

Można obliczyć prawdopodobieństwo dla każdego możliwego rozdania.

Można sobie wyobrazić, że ma się do czynienia w wielką kostką na początku gry.

Idea: Oblicz wartość minimax dla każdej akcji w każdym rozdaniu, potem wybierz akcję z największą oczekiwanią wartością ze wszystkich rozdań.

Przypadek szczególny: Jeżeli akcja jest optymalna dla wszystkich rozdań, to jest optymalna.

GIB- program do gry w brydża stosuje w przybliżeniu tą koncepcję
(1) generuje 100 rozdań zgodnych z licytacją
(2) wybiera akcję, która wygra więcej lewych

Podsumowanie

Gry są interesujące i przenoszą się na zagadnienia rzeczywiste — rozwiązywanie problemów podejmowania decyzji.

Prezentują kilka interesujących cech SI:

- Doskonałość jest nieosiągalna dlatego stosuje się przybliżenia.
- Im więcej danych o problemie jesteśmy w stanie zebrać tym łatwiej rozwiązać zadanie np. zbudować dobrą funkcję szacującą.
- Niepewność ogranicza wyznaczanie wartości dla stanów.
- Optymalne decyzje zależą od stanu informacji nie od stanu rzeczywistego.