Algorytm MIN-MAX — historia, odmiany, usprawnienia

Przemysław Klęsk
pklesk@wi.zut.edu.pl

Katedra Metod Sztucznej Inteligencji i Matematyki Stosowanej

Spis treści

Spis treści

Teoria gier

Teoria gier

Gałąź matematyki zajmująca się zachowaniami w sytuacjach konfliktowych (strategicznych), gdzie wynik uczestnika, zależy od wyborów podejmowanych przez niego i innych. Bywa nazywana *teorią racjonalnych zachowań*. Oprócz informatyki stosowana w naukach socjologicznych, ekonomicznych, militarnych (historycznie wcześniej).

Ważne historycznie prace:

- Émil Borel, *Aplikacje dla gier losowych* (fr. *Applications aux Jeux de Hasard*), 1938.
- John von Neuman i Oskar Morgenstern, *Teoria gier i zachowań ekonomicznych* (ang. *Theory of Games and Economic Behavior*), 1944.

Pojęcia

Gra

Pewna sytuacja konfliktowa, gdzie:

- można wskazać co najmniej dwóch graczy,
- każdy gracz ma do wyboru pewną liczbę możliwych *strategii*, określających dokładnie sposób rozgrywania przez niego gry,
- wynik gry jest jednoznacznie określony przez kombinację strategii wybranych przez graczy.

Strategia

Pełny zestaw decyzji (o wyborach, ruchach), które gracz ma podjąć, dla wszystkich możliwych do zastania stanów gry.

Strategia jest często niemożliwa do spisania/przechowania, ze względu na swój rozmiar.

Pojęcia

Gra skończona

Gra, w której istnieje gwarancja, że gra ulegnie zakończeniu.

Gra o sumie zerowej

Gra, w której *wypłaty* wszystkich graczy (zdeterminowane przez wynik gry), sumują się do zera.

Dla szachów konwencja: 0, 1, $\frac{1}{2}$ (suma zero powstaje przez przeskalowanie: $2x - 1$).

Twierdzenie o minimaksie

Twierdzenie o minimaksie (von Neuman, 1928)

Dla każdej skończonej gry dwuosobowej o sumie zero, istnieje przynajmniej jedna optymalna *strategia mieszana*. Tym samym istnieje wartość gry v, taka że gracz pierwszy stosując optymalną strategię gwarantuje sobie oczekiwaną wypłatę niegorszą niż v, a gracz drugi stosując optymalną strategię gwarantuje sobie wypłatę niegorszą niż $-v$.

Dla gier o sumie zerowej rozwiązanie minimaksowe jest tożsame z *równowagą Nasha* (szersze pojęcie).

Przykład dla twierdzenia

	B wybiera b_1	B wybiera b_2	B wybiera b_3
A wybiera a_1	3	−2	3
A wybiera a_2	−1	0	4
A wybiera a_3	−4	−3	1

- Macierz wypłat $W = \{w_{ij}\}$ dla gry o sumie zerowej, gdzie gracze A i B wykonują **jednoczesne ruchy**.
- Co jest minimaksowym wyborem dla A i B?
- Czy jest to rozwiązanie stabilne?
- Czy istnieją wybory zdominowane?

Przykład dla twierdzenia

	B wybiera b_1	B wybiera b_2	B wybiera b_3
A wybiera a_1	3	-2	3
A wybiera a_2	-1	0	4
A wybiera a_3	-4	-3	1

- Minimaksowym wyborem dla A jest a_2, ponieważ najgorszym możliwym wynikiem dla A jest wówczas -1:
$$\max_i \min_j w_{ij} = -1. \tag{1}$$
- Minimaksowym wyborem dla B jest b_2, ponieważ najgorszym możliwym wynikiem dla B jest wówczas 0:
$$\min_j \max_i w_{ij} = 0. \tag{2}$$
- Rozwiązanie (a_2, b_2) jest **niestabilne**, ponieważ jeżeli B wierzy, że A wybierze a_2, to B wybierze b_1, aby dostać wypłatę -1; wówczas, jeżeli A wierzy, że B wybierze b_1, to A wybierze a_1, aby dostać wypłatę 3, itd.
- **Wybory zdominowane**: a_3 i b_3 — niezależnie od wyboru przeciwnika, pozostałe nasze wybory są lepsze (dokładnie: niegorsze). Stąd, macierz można zredukować wykreślając trzeci wiersz i trzecią kolumnę.

Przykład dla twierdzenia

	B wybiera b_1	B wybiera b_2
A wybiera a_1	3	-2
A wybiera a_2	-1	0

$$\begin{pmatrix} p & 1-p \end{pmatrix} \cdot W \cdot \begin{pmatrix} q \\ 1-q \end{pmatrix} = -q - 2p + 6pq. \tag{3}$$

- **Strategia mieszana** to kombinacja wyborów — strategii czystych — z określonymi prawdopodobieństwami (częstościami).
- Obecność termu pq wskazuje na istnienie **punktu siodłowego**.
- Żądając, aby $\frac{\partial \cdot}{\partial p} = 0$ i $\frac{\partial \cdot}{\partial q} = 0$, otrzymujemy rozwiązanie:

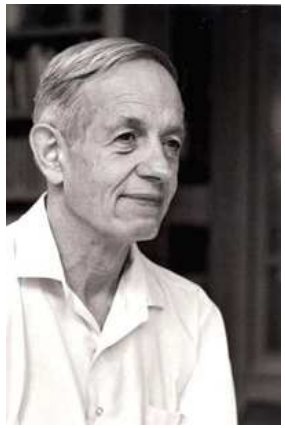
$$p = \frac{1}{6}, \quad q = \frac{1}{3}. \tag{4}$$

- **Wartość gry**: $v = -\frac{1}{3}$.
- Formalnie, jeżeli P i Q reprezentują strategie mieszane (wektory prawdopodobieństw), to:

$$\max_P \min_Q P^T \cdot W \cdot Q = \min_Q \max_P P^T \cdot W \cdot Q = v. \tag{5}$$

- Jeżeli istnieje więcej niż jedna optymalna strategia mieszana, to istnieje ich nieskończenie wiele.

Równowaga Nasha (NEQ)

John Nash, ur. 1928, laureat nagrody Nobla w 1994 (z ekonomii).

Nieformalnie

W grze wieloosobowej, mówimy że pewien zestaw strategii poszczególnych graczy stanowi *równowagę Nasha*, wtedy i tylko wtedy, gdy każda strategia wśród nich stanowi najlepszą odpowiedź na wszystkie pozostałe i żaden gracz nie może zyskać poprzez zmianę swojej strategii przy ustalonych pozostałych.

Równowaga Nasha (NEQ)

Formalnie

- W grze o n graczach, niech S_i oznacza zbiór możliwych strategii dla gracza i.
- Niech S oznacza **przestrzeń wszystkich strategii** jako produkt kartezjański zbiorów strategii poszczególnych graczy:

$$S = S_1 \times S_2 \times \cdots \times S_n$$

- Dla pewnego zestawu strategii $(s_1, \ldots, s_n)$ pochodzących od poszczególnych graczy, niech $W_i(s_1, \ldots, s_n)$ określa **wypłatę** i-tego gracza. Zatem W_i jako funkcja jest odwzorowaniem:

$$W_i \colon S \to \mathbb{R}.$$

  Niech W oznacza funkcję wektorową: $W = (W_1, \ldots, W_n)$.
- **Grę** można rozumieć jako parę (S, W).
- Mówimy, że zestaw strategii $(s_1^*, \ldots, s_n^*)$ stanowi **równowagę Nasha**, wtedy i tylko wtedy, gdy:

$$\forall i \quad \forall s_i \in S_i : \qquad W_i(s_1^*, \ldots, s_i^*, \ldots, s_n^*) \geqslant W_i(s_1^*, \ldots, s_i, \ldots, s_n^*). \tag{6}$$

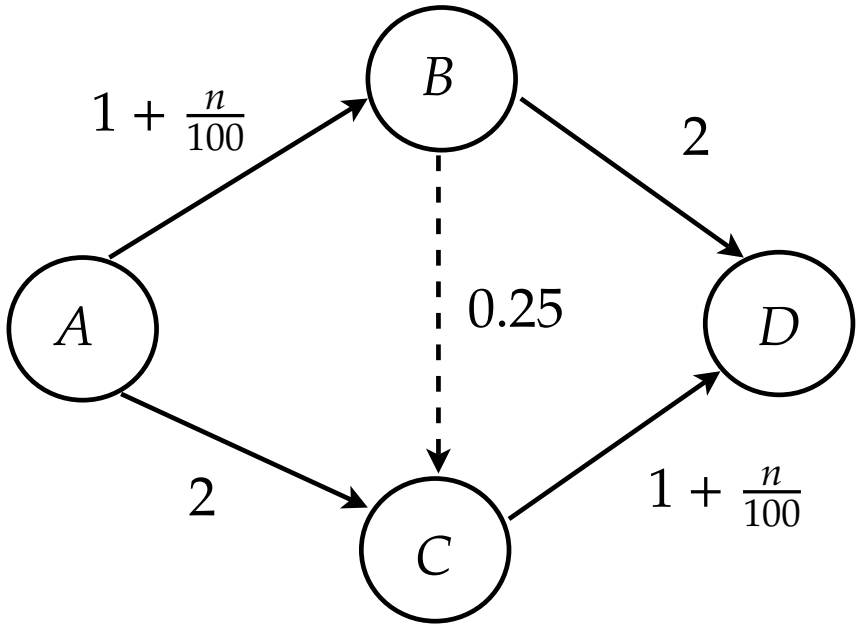
Równowaga Nasha (NEQ)

- Innym sposobem wypowiedzenia NEQ, jest to, że s_i^* jest rozwiązaniem:

$$\max_{s_i \in S_i} W_i(s_1^*, \ldots, s_i, \ldots, s_n^*) \tag{7}$$

  dla wszystkich i.
- Idea NEQ jest stosowana do analizy/przewidywania co stanie się, gdy kilka osób/instytucji musi podjąć równocześnie decyzję, i gdy wynik zależy od wszystkich decyzji. Wyniku nie można przewidzieć, analizując decyzje graczy oddzielnie (w izolacji).
- NEQ nie musi koniecznie oznaczać najlepszego grupowego (sumarycznego) wyniku z gry i może wydawać się nieracjonalna z punktu widzenia osób poza grą (np. *dylemat więźnia*, *paradoks Braess'a*).
- W wielu przypadkach gracze mogliby poprawić sumaryczny wynik, gdyby uzgodnili strategie różne od NEQ (np. kartele biznesowe zamiast konkurencji).

Paradoks Braess'a



Zadanie

Zakładając samolubność i racjonalność kierowców, znaleźć oczekiwany przepływ ruchu ulicznego (równowagę Nasha) dla 100 kierowców podróżujących z A do D w dwóch przypadkach: (1) gdy krawędź BC nie istnieje, (2) gdy krawędź BC istnieje.
Potraktować problem jako grę, gdzie każdy gracz ma odpowiednio 2 lub 3 strategie: ABD, ACD i ewentualnie $ABCD$. Wypłatą jest czas podróży wybraną drogą. Dla krawędzi AB i BD, n oznacza liczbę graczy, którzy wybrali tę krawędź jako fragment drogi.

Paradoks Braess'a — rozwiązanie

p, q, r — liczba kierowców, którzy wybrali odpowiednio strategie: ABD, ACD, $ABCD$.

Przypadek 1

$$\begin{cases} 1 + \frac{p}{100} + 2 & = 1 + \frac{q}{100} + 2; \\ p + q & = 100. \end{cases} \tag{8}$$

Rozwiązanie: $p = q = 50$, koszt drogi (wartość gry) $v = 3.5$.

Paradoks Braess'a — rozwiązanie

Przypadek 2

$$\begin{cases} 1 + \frac{p+r}{100} + 2 & = 1 + \frac{q+r}{100} + 2 = 1 + \frac{p+r}{100} + 0.25 + 1 + \frac{q+r}{100}; \\ p + q + r & = 100. \end{cases} \tag{9}$$

Rozwiązanie: $p = q = 25$, $r = 50$, koszt drogi (wartość gry) $v = 3.75$.

Kierowcy jechaliby krócej, gdyby umówili się, że nie używają BC.

Spis treści

Drzewo gry

Dla pewnego stanu s przyjmijmy, że istnieje n możliwych wyborów: $a_1, a_2, \ldots, a_n$, powodujących powstanie ze stanu s nowych stanów gry, odpowiednio: $s_1, s_2, \ldots, s_n$. Dalej, dla każdego z tych stanów mamy znowu pewien zestaw możliwych wyborów. Kontynuując to postępowanie w naturalny sposób otrzymujemy *strukturę drzewa*.

Problemy przeszukiwania

- **zbyt duża przestrzeń przeszukiwań** — gdy stworzenie pełnego drzewa niemożliwe lub zbyt kosztowne pamięciowo/obliczeniowo (problem wykładniczy).
- **gry o niepełnej informacji** — gracze nie posiadają pełnej informacji o stanie gry (np. karty trzymane przez przeciwnika, kostki przeciwnika w Rummikub, literki w Scrabble).
- **czynnik losowy** — gdy elementem gry są generatory losowe (np. rzut kostką, tasowanie talii kart).

Mierniki złożoności gier

1. **Złożoność przestrzeni stanów** (ang. *state-space complexity*) — liczba dozwolonych pozycji w grze osiągalnych poczynając od pozycji początkowej. Bywa szacowana z góry, gdy łatwiej w rachunku uwzględnić nielegalne pozycje.
2. **Rozmiar drzewa gry** (ang. *game tree size*) — liczba możliwych do rozegrania różnych gier; lub równoważnie, liczba liści w drzewie z korzeniem ustalonym na pozycję początkową. Uwzględnia się tu możliwość powtarzania się w drzewie tych samych stanów gry osiąganych w wyniku tych samych ruchów, ale wykonanych w różnym porządku. Bywa szacowana z góry dopuszczając rozrost drzewa w wyniku nielegalnych ruchów.

Mierniki złożoności gier

3. **Złożoność decyzyjna** (ang. *decision complexity*) — liczba liści w najmniejszym *drzewie decyzyjnym*, które ustala ocenę pozycji początkowej.
4. **Złożoność drzewa gry** (ang. *game tree complexity*) — liczba liści w najmniejszym *drzewie decyzyjnym o pełnej szerokości*, które ustala ocenę pozycji początkowej. Takie drzewo uwzględnia wszystkie możliwości decyzyjne dla obu graczy. Odpowiada liczbie potrzebnych operacji w przeszukiwaniu minimaksowym.

Drzewo decyzyjne (z punktu widzenia pierwszego gracza)

Poddrzewo drzewa gry, w którym wszystkie stany mają etykiety: *zwycięstwo*, *remis*, *porażka*.

- Stan uzyskuje etykietę *zwycięstwo*, gdy *dowolny* stan potomny ma etykietę *zwycięstwo*.
- Stan uzyskuje etykietę *porażka*, gdy *wszystkie* stany potomne mają etykietę *porażka*.
- Stan uzyskuje etykietę *remis*, gdy przynajmniej jeden stan potomny ma etykietę *remis*, a wszystkie pozostałe *porażka*.

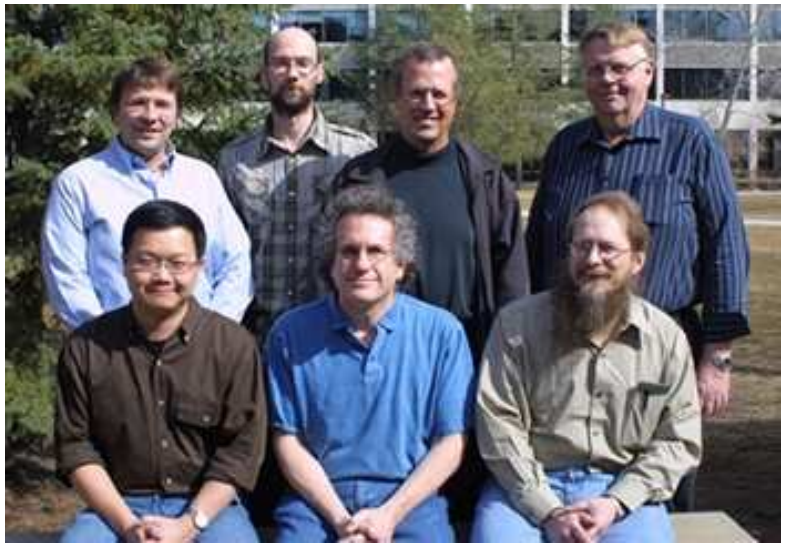
Drzewo decyzyjne uwzględnia wszystkie możliwości dla każdej decyzji drugiego gracza, i tylko jedną możliwość (najlepszą) dla każdej decyzji pierwszego gracza.

Szacunki złożoności dla niektórych gier

gra	rozmiar planszy (liczba pól)	złożoność przestrzeni stanów	złożoność drzewa gry
kółko i krzyżyk	9	10^3	10^5
connect 4	42	10^{13}	10^{21}
warcaby angielskie	32	10^{20}	10^{31}
hex także: *Nash* lub *John*	121	10^{56}	?
szachy	64	10^{47}	10^{123}
connect 6	361	10^{172}	10^{140}
backgammon także: *tryk-trak*	28	10^{20}	10^{144}
Go także: *batuk*	361	10^{171}	10^{360}

Projekt Chinook (warcaby)

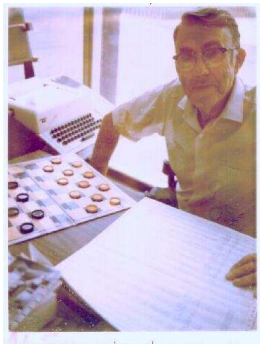
- Dla *warcab angielskich* (8×8, damki ruszają się w dowolnym kierunku ale o 1 pole).
- Projekt rozpoczęto w 1989. **Cel: pokonanie człowieka w mistrzostwach świata w warcabach.**
- **Autorzy:** (tył od lewej) Yngvi Bjørnsson, Neil Burch, Rob Lake, Joe Culberson (przód od lewej) Paul Lu, Jonathan Schaeffer, Steve Sutphen. **Uniwersytety:** Rejkjavik (Islandia), Alberta (Kanada), Hakodate (Japonia).



Projekt Chinook (warcaby)

- W 1990 uznano prawo programu do udziału w mistrzostwach i grze przeciwko człowiekowi.
- Program przegrał mistrzostwa w 1992, ale wygrał w 1994. W 1996 wycofano możliwość udziału Chinooka (program silniejszy niż którykolwiek człowiek).
- Przestrzeń przeszukiwań rzędu: $5 \cdot 10^{20}$. Baza danych z ewidencją wielu stanów gry i informacją o najlepszym posunięciu w danym stanie.
- **29.04.2007 r. autorzy projektu ogłosili warcaby angielskie za rozwiązane!** Czarne (rozpoczynające grę) mają gwarancję remisu przy doskonałej grze. Białe mają również gwarancję remisu, niezależnie od pierwszego posunięcia czarnych.
- Po dziś dzień warcaby, to największa rozwiązana gra.

Projekt Chinook — dziedzictwo Samuela



- **Artur Samuel** napisał silnik programu w 1950 r. w ramach projektu sponsorowanego przez IBM.
- W 1952 r. dodano **genetyczny element samouczenia** — dwa programy grające przeciw sobie, z powtórzeniami. Wyewoluowany program pokonywał amatorów i średniozaawansowanych graczy.
- Po prezentacji programu w 1956 r. udziałowcom IBM, akcje IBM skoczyły o 15 punktów procentowych.
- W 1962 r. program rozegrał upubliczniony mecz przeciwko Robertowi Nealy (niewidomy mistrz warcabowy), w którym odniósł zwycięstwo (mocno rozreklamowane). Nealy nie był graczem „z najwyższej półki”.
- **W efekcie projektu rozpowszechniło się fałszywe przekonanie, że warcaby zostały rozwiązane na tamten moment.** Bjørnsson miał z tego tytułu problemy z uzyskaniem finansowania dla Chinooka w latach 80-ych.
- Rok później program przegrał rewanż: 1 porażka, 5 remisów. W 1966 program przegrał 8 kolejnych gier przeciwko lepszym graczom: Derekowi Oldbury, Walterowi Hellman’owi.

Spis treści

Algorytm MIN-MAX (lub minimaks)

Procedura mmEvaluateMax(s, d, D)

1. Jeżeli s jest terminalem, to zwróć $h(s)$.
2. Przypisz $v = -\infty$.
3. Dla wszystkich stanów t będących potomkami s wykonuj:
   1. $v := \max\{v, \text{mmEvaluateMin}(t, d + \frac{1}{2}, D)\}$.
4. Zwróć v.

Procedura mmEvaluateMin(s, d, D)

1. Jeżeli s jest terminalem, to zwróć $h(s)$.
2. Przypisz $v = \infty$.
3. Dla wszystkich stanów t będących potomkami s wykonuj:
   1. $v := \min\{v, \text{mmEvaluateMax}(t, d + \frac{1}{2}, D)\}$.
4. Zwróć v.

Algorytm *przycinanie α-β* (ang. *α-β cut-offs* lub *α-β pruning*)

- Wielu niezależnych odkrywców: Samuel (1952), McCarthy (1956), Newell i Simon (1958).
- W trakcie analizy drzewa propagowane są w dół i górę drzewa wartości:
  α — **gwarantowana dotychczas wypłata gracza maksymalizującego**,
  β — **gwarantowana dotychczas wypłata gracza minimalizującego**.
- W wywołaniu dla korzenia zadaje się $\alpha = -\infty$, $\beta = \infty$.
- Dzieci (i ich poddrzewa) są analizowane dopóki $\alpha < \beta$.
- W momencie gdy $\alpha \geqslant \beta$, przestajemy rozpatrywać kolejne dzieci (i ich poddrzewa) — nie będą one miały wpływu na wynik całego drzewa, są wynikiem nieoptymalnego postępowania graczy.
- W optymistycznym przypadku **zysk w złożoności** względem MIN-MAX'a z $O(b^D)$ na $O\left(b^{D/2}\right) = O\left(\sqrt{b^D}\right)$, gdzie b — *branching factor* (stały lub średni współczynnik rozgałęziania). Np. dla szachów $b \approx 40$.
- Dzięki zyskowi w złożoności można szukać głębiej.

Algorytm *przycinanie* α-β (wersja *fail-hard* zwraca wynik z przedziału $[\alpha, \beta]$)

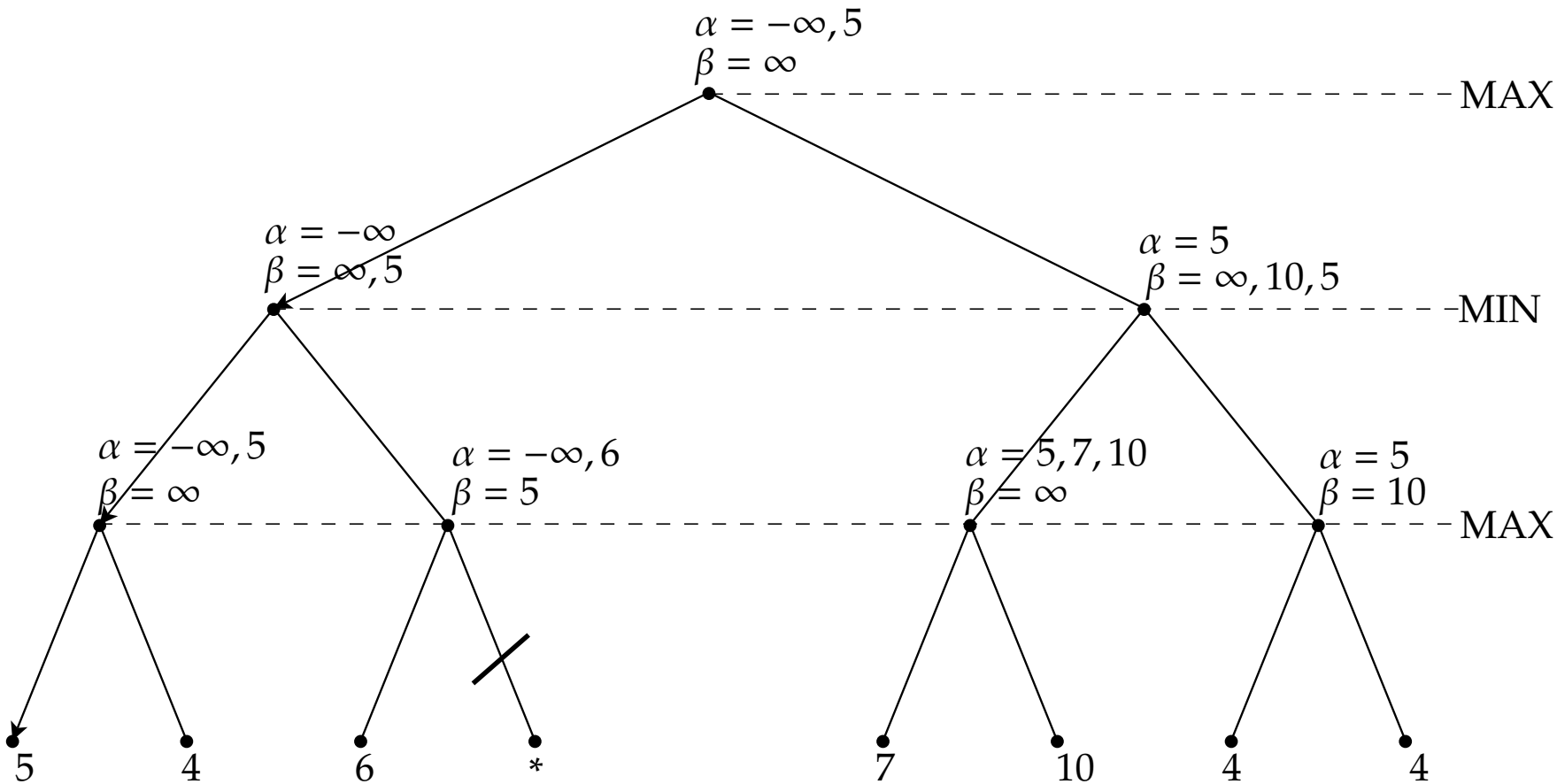
Procedura alphaBetaEvaluateMax(s, d, D, α, β)

1. Jeżeli s jest terminalem, to zwróć $h(s)$.
2. Dla wszystkich stanów t będących potomkami s wykonuj:
   1. $v :=$ alphaBetaEvaluateMin$(t, d + \frac{1}{2}, D, \alpha, \beta)$.
   2. Jeżeli $\beta \leqslant v$, to zwróć β. (przycięcie)
   3. $\alpha := \max\{\alpha, v\}$.
3. Zwróć α.

Procedura alphaBetaEvaluateMin(s, d, D, α, β)

1. Jeżeli s jest terminalem, to zwróć $h(s)$.
2. Dla wszystkich stanów t będących potomkami s wykonuj:
   1. $v :=$ alphaBetaEvaluateMax$(t, d + \frac{1}{2}, D, \alpha, \beta)$.
   2. Jeżeli $v \leqslant \alpha$, to zwróć α. (przycięcie)
   3. $\beta := \min\{\beta, v\}$.
3. Zwróć β.

Ilustracja *przycinania* α-β — przykład 1



Dlaczego optymistyczna złożoność $O(b^{D/2})$?

- W tradycyjnym MIN-MAX:

$$O(\underbrace{b \cdot b \cdots b}_{D\text{-krotnie } b}) = O(b^D). \tag{10}$$

- W α-β przy parzystej liczbie poziomów, **optymistycznie** mamy:

$$O(\underbrace{b \cdot 1 \cdot b \cdot 1 \cdots b \cdot 1}_{D/2\text{-krotnie } b}) = O\left(b^{D/2}\right). \tag{11}$$

Wyjaśnienie: potrzebujemy zbudować wszystkie możliwe dzieci dla pierwszego gracza, ale **zakładamy, że ruchu są optymalnie posortowane** i w związku z tym w każdym dziecku już pierwszy ruch drugiego gracza pozwoli odrzucić wszystkie dalsze ruchy drugiego gracza (zajdzie $\alpha \geqslant \beta$) jako nieoptymalne. I tak dalej rekurencyjnie.

- Istnieją oszacowania dla przypadku średniego (losowe posortowanie), dające $O(b^{\frac{3}{4}D})$.
- W szachach dla: $b = 40$ i $D = 12$ (12 półruchów), stosunek liczby odwiedzonych stanów przy pesymistycznym sortowaniu do liczby odwiedzonych stanów przy optymistycznym sortowaniu jest 40^6 czyli rzędu 10^9.

Algorytm *przycinanie* α-β (wersja *fail-soft*, wynik nie musi należeć do przedziału $[\alpha, \beta]$)

Procedura fsAlphaBetaEvaluateMax(s, d, D, α, β)

1. Jeżeli s jest terminalem, to zwróć $h(s)$.
2. Dla wszystkich stanów t będących potomkami s wykonuj:
   1. $v :=$ fsAlphaBetaEvaluateMin$(t, d + \frac{1}{2}, D, \alpha, \beta)$.
   2. $\alpha := \max\{\alpha, v\}$.
   3. Jeżeli $\alpha \geqslant \beta$, to zwróć α. (przycięcie)
3. Zwróć α.

Procedura fsAlphaBetaEvaluateMin(s, d, D, α, β)

1. Jeżeli s jest terminalem, to zwróć $h(s)$.
2. Dla wszystkich stanów t będących potomkami s wykonuj:
   1. $v :=$ fsAlphaBetaEvaluateMax$(t, d + \frac{1}{2}, D, \alpha, \beta)$.
   2. $\beta := \min\{\beta, v\}$.
   3. Jeżeli $\alpha \geqslant \beta$, to zwróć β. (przycięcie)
3. Zwróć β.

Twierdzenie Knuth'a-Moore'a (1975)

Praca: **Knuth D.E., Moore R.W., *„An Analysis of Alpha-Beta Pruning"*, Artificial Intelligence, 1975.**

„Twierdzenie o oknie α-β"

Niech v^* oznacza prawdziwy (dokładny) wynik gry otrzymany procedurą MIN-MAX. Niech v oznacza wynik procedury fsAlphaBeta (fail-soft) uruchomionej na rzecz korzenia z parametrami początkowymi α, β. Wówczas, możliwe są trzy sytuacje:

1. $\alpha < v < \beta \quad \Rightarrow \quad v = v^*$,
2. $v \leqslant \alpha$ *(failing low)* $\quad \Rightarrow \quad v^* \leqslant v \quad$ *(v stanowi górne ograniczenie na v^*)*,
3. $\beta \leqslant v$ *(failing high)* $\quad \Rightarrow \quad v \leqslant v^* \quad$ *(v stanowi dolne ograniczenie na v^*).*

Jedna z konsekwencji: *fsAlphaBeta*(korzeń, $-\infty, \infty$) $= v^*$.

Twierdzenie przydaje się do zbudowania zaawansowanych algorytmów przeszukiwań: *Negascout*, *MTD-f* opartych o tzw. *zerowe okna przeszukiwań*.

Algorytm *Quiescence*

- Próbuje naśladować intuicję ludzi w grach, poprzez: **dalsze rozwijanie drzewa i analizę dla liści *głośnych*, a nie rozwijanie drzewa dla liści *cichych* (tylko od razu zwrot heurystyki)**.
- Częściowo rozwiązuje problem *efektu horyzontu*.
- **Cichą** nazywamy sytuację, gdzie nie występują gwałtowne zmiany w wartości funkcji heurystycznej pomiędzy danym stanem a rodzicem (np. bicia).
- Ocena czy sytuacja jest cicha czy nie, może nie być łatwa i może wymagać heurystyki. Ważne, aby była szybka i kosztowała mniej niż rozwinięcie kolejnego poziomu w drzewie.
- Algorytm *Quiescence* niekoniecznie musi być zastosowany na poziomie liści, a już wcześniej. Elementem heurystyki oceniającej cichość może być głębokość w drzewie tj. **im głębiej tym większa chęć do pozostawienia cichych stanów nierozwiniętych**.
- Dokładny opis np. u: D. Laramée, *Chess Programming Part V: Advanced Search*, 2000.

Sortowanie stanów-dzieci w *przycinaniu α-β*

Warto sortować ruchy, zwłaszcza „u góry drzewa”, gdzie jest to względnie tanie (bo mniej stanów na wyższych poziomach), a może skutkować sporą oszczędnością głębiej.

Heurystyki sortujące

- w szachach: ruchy ***„zbijające najpierw”***,
- w wielu grach karcianych np. w brydżu: karty ***„skrajne najpierw, środkowe później”***; np. układ $A, D, 8, 6, 5, 2$ posortować do: $A, 2, D, 5, 8, 6$; można też porządek ustalić na podstawie pozycji gracza w ramach jednej lewy (np. gracz ostatni zwykle wysoka karta najpierw, gracz drugi zwykle niska najpierw, itp.),
- ***sortowanie wg heurystyki pozycyjnej*** — oceń i posortuj od razu dzieci za pomocą heurystyki h pozycję, przed uruchomieniem rekurencji w dół.

Sortowanie stanów-dzieci w *przycinaniu α-β*

Heurystyki sortujące (ciąg dalszy)

- ***„refutation table"*** (pol. *tablica odrzucenia*) — tablica zapamiętująca ruchy (*refutation moves*), które spowodowały odcięcia, zwykle dla płytkich głębokości, co pozwala rozpatrywać te ruchy w pierwszej kolejności w kolejnych iteracjach algorytmu — (tzw. *progressive search* lub *iterative search*). Historycznie, pierwsze programy szachowe z dużymi ograniczeniami na pamięć trzymały zwykle b ruchów odrzucających będących ruchami w dzieciach korzenia korzenia, lub ewentualnie $b + b^2$ biorąc poziom dalej. Nazywana też *tablicą najlepszych kontynuacji*.
- ***„killer heuristic"*** — przechowywanie dla każdego poziomu krótkiej listy ruchów *killer moves* (zwykle 2, 3), które spowodowały α-β przycięcie i rozważanie tych ruchów w pierwszej kolejności dla innych stanów na tym poziomie oraz w kolejnych iteracjach; intuicyjne uzasadnienie: jeżeli ruch był dobry w pewnym stanie, to być może jest on też dobry w stanie do niego podobnym.
  Jeżeli ruch *non-killer* spowodował przycięcie, to jest on wprowadzany na listę *killer moves* w miejsce „najsłabszego" z nich.

Tablica transpozycji

- Nazwa wywodzi się z szachów i oznacza możliwość otrzymania tej samej pozycji (stanu) poprzez różne sekwencje ruchów.
- Jeżeli rekurencja wgłąb dla takiej pozycji jest już przeliczona, to można zaoszczędzić czas biorąc gotowy wynik.
- Implementowana jako *mapa haszująca* (szybkość, analogia do zbioru *Closed* odwiedzonych stanów w A^*, BFS, itp.). Kluczami w mapie są stany bądź ich skróty (np. w szachach tylko pozycje maks. 32 bierek plus informacje o roszadach i biciach *en passant*).
- Warunki wykorzystania stanu z tablicy transpozycji:
  - Głębokość stanu w tablicy niewiększa niż stanu nowego (aby wiedzieć, że gotowa ocena pochodzi z analizy poddrzewa o takim samym lub większym rozmiarze).
  - Sprawdzenie, czy przycięcia α-β nie sprawiły, że odwiedzony stan nie był zbadany wystarczająco głęboko a teraz nowy stan powinien być (widełki α-β dla stanu w tablicy powinny być niewęższe).
- Bywa stosowana jako książka debiutów lub końcówek (szachy, warcaby).

Algorytm *Scout* (pol. *Zwiadowca*)

- Historycznie pomysł **J. Pearl**'a (1980): można „zwiadowczo" i mniej kosztownie badać, czy dotychczasową wypłatę można poprawić. Dwie rekurencyjne procedury eval(·) i test(·), ta druga zwracała wartość logiczną, czy możliwa poprawa.
- Pomysł wykorzystał i udoskonalił **A. Reinefeld**, *„An Improvement to the Scout Tree Search Algorithm"*, ICCA Journal, (1983), wprowadzając tzw. ***zerowe okna*** α-β (ang. *null window*, *zero window*, *scout window*).
- Jeżeli wypłaty w grze są tylko całkowitoliczbowe, to *zerowe okno* ma miejsce wtedy, gdy

$$\alpha + 1 = \beta. \tag{12}$$

- Pomysł „współpracuje" z twierdzeniem Knuth'a-Moore'a.

Algorytm *Scout*

Definicja

Mówimy, że zadane okno α-β ***poskutkowało*** (ang. *succeeded*), jeżeli wartość v zwrócona przez procedurę *fsAlphaBeta* (fail-soft) jest taka że: $\alpha < v < \beta$. Wynika z tego (Knuth-Moore), że prawdziwa wartość gry v^* jest równa v.

Definicja

Mówimy, że zadane okno α-β ***zawiodło dołem*** (ang. *failed low*), jeżeli wartość v zwrócona przez procedurę *fsAlphaBeta* (fail-soft) jest taka że: $v \leqslant \alpha$. Wynika z tego (Knuth-Moore), że v jest górnym oszacowaniem wartości gry: $v^* \leqslant v$.

Definicja

Mówimy, że zadane okno α-β ***zawiodło górą*** (ang. *failed high*), jeżeli wartość v zwrócona przez procedurę *fsAlphaBeta* (fail-soft) jest taka że: $\beta \leqslant v$. Wynika z tego (Knuth-Moore), że v jest dolnym oszacowaniem wartości gry: $v \leqslant v^*$.

Algorytm *Scout*

- **Im zadane okno jest węższe, tym większa szansa na powstawanie odcięć.**
- Tylko pierwsze dziecko każdego stanu jest analizowane z pełnym oknem α-β, **drugie dziecko i dalsze dzieci są analizowane zerowym oknem**, tj. α-$(\alpha + 1)$ lub $(\beta - 1)$-β odpowiednio dla stanów typu MAX, MIN.
- **Zerowe okno musi zawieźć w jedną ze stron.**
- Jeżeli zerowe okno nastawione dla dziecka stanu MAX zawiedzie dołem, to nie przejmujemy się — wypłaty gracza maksymalizującego nie można było poprawić w ramach tego dziecka (**zysk obliczeniowy**, prawdopodobna większa liczba odciąć w ramach poddrzewa tego dziecka).
- Jeżeli zerowe okno nastawione dla dziecka stanu MAX, zawiedzie górą, to musimy przeszukiwanie drzewa dla tego dziecka powtórzyć (**strata obliczeniowa**) z szerszym oknem v-β, aby uzyskać dokładną wartość poddrzewa dziecka. Uwaga: i tak jest to okno węższe niż pełne α-β.
- Powyższe dwie uwagi odpowiednio odwrotne dla węzłów MIN.

Algorytm *Scout*

Procedura scoutMax(s, d, D, α, β)

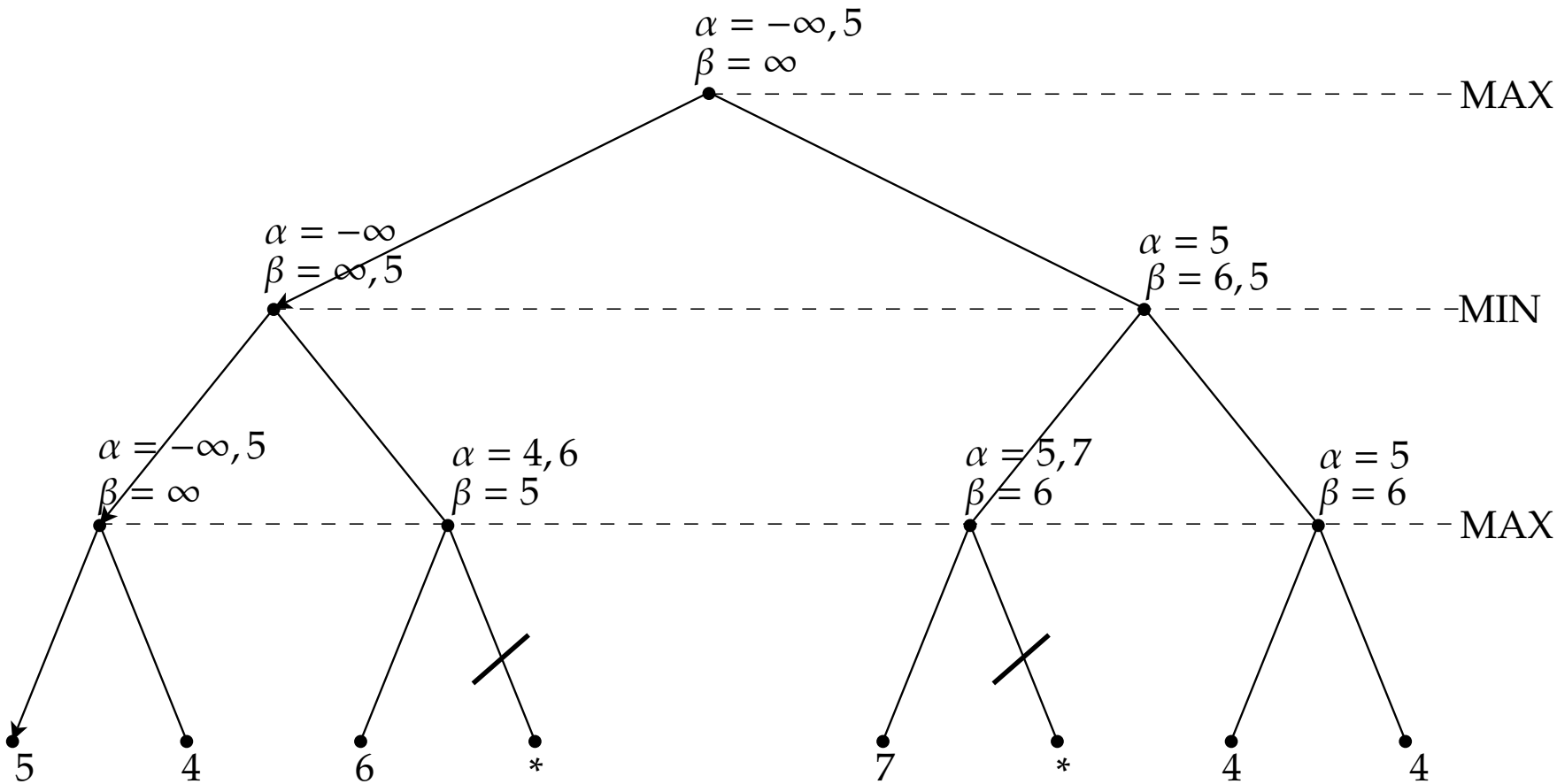
1. Jeżeli s jest terminalem, to zwróć $h(s)$.
2. $b := \beta$.
3. Dla wszystkich stanów t będących potomkami s wykonuj:
   1. $v :=$ scoutMin$(t, d + \frac{1}{2}, D, \alpha, b)$.
   2. Jeżeli t nie jest pierwszym dzieckiem i $D - d > 2 \cdot \frac{1}{2}$ i $b \leqslant v$ (failing high), to:
      1. $v :=$ scoutMin$(t, d + \frac{1}{2}, D, v, \beta)$. (ponowne przeszukanie szerszym oknem)
   3. $\alpha := \max\{\alpha, v\}$.
   4. Jeżeli $\alpha \geqslant \beta$, to zwróć α. (przycięcie)
   5. $b := \alpha + 1$.
4. Zwróć α.

Algorytm *Scout*

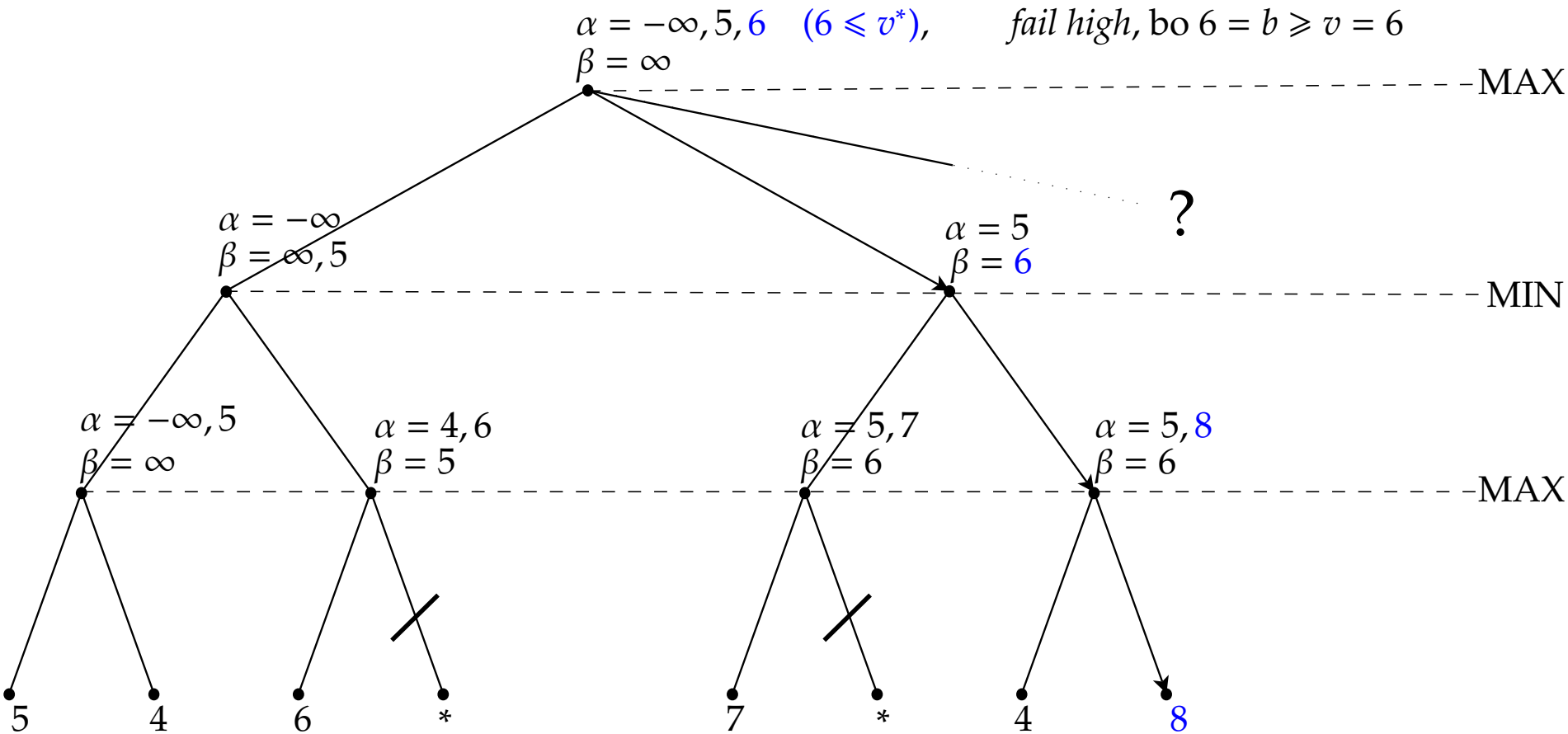
Procedura scoutMin(s, d, D, α, β)

1. Jeżeli s jest terminalem, to zwróć $h(s)$.
2. $a := \alpha$.
3. Dla wszystkich stanów t będących potomkami s wykonuj:
   1. $v :=$ scoutMax$(t, d + \frac{1}{2}, D, a, \beta)$.
   2. Jeżeli t nie jest pierwszym dzieckiem i $D - d > 2 \cdot \frac{1}{2}$ i $v \leqslant a$ (failing low), to:
      1. $v :=$ scoutMax$(t, d + \frac{1}{2}, D, \alpha, v)$. (ponowne przeszukanie szerszym oknem)
   3. $\beta := \min\{\beta, v\}$.
   4. Jeżeli $\alpha \geqslant \beta$, to zwróć β. (przycięcie)
   5. $a := \beta - 1$.
4. Zwróć β.

Ilustracja *Scout'a* — przykład 1



Ilustracja *Scout'a* — przykład 2



Algorytm *Scout*

- Warunek $D - d > 2 \cdot \frac{1}{2}$ sprawdza, czy jesteśmy na głębokości 2 półruchy do końca lub głębszej. Jeżeli tak, to niepotrzebne jest powtórne przeszukanie pomimo sytuacji *fail*, bo algorytm działa dokładnie na tych głębokościach.
- Algorytm dobrze sprawdza się w *progressive search* we współpracy z heurystykami sortującymi ruchy, szczególnie z *killer heuristic*, gdzie najlepszy ruch (i najlepsza ścieżka) jest zwykle rozpatrywany jako pierwszy. Z tego powodu bywa nazywany ***Principal Variation Search***.
- Doświadczenia wskazują, że **zyski obliczeniowe wynikające z zerowych okien i częstszych odcięć są zwykle większe niż straty wynikające z powtórnych dokładnych przeszukań**.
- Doświadczenia Reinefelda pokazały, że dla drzew o rozgałęzianiu $b \in [20, 60]$ (np. szachy), *Scout* odwiedzał około 20% mniej liści. Testy dla głębokości $4, 5$ półruchów.

Algorytm *Negamax*

- Fakt:

$$\forall n \in \mathbb{N} \quad \forall a_1, a_2, \ldots, a_n \qquad \min\{a_1, a_2, \ldots, a_n\} = -\max\{-a_1, -a_2, \ldots, -a_n\}.$$

- Dzięki temu można uprościć implementację zastępując bliźniacze procedury rekurencyjne jedną procedurą we wszystkich algorytmach: *Negamax* (tak naprawdę *negaAlphaBeta*), *Negascout*, itp.

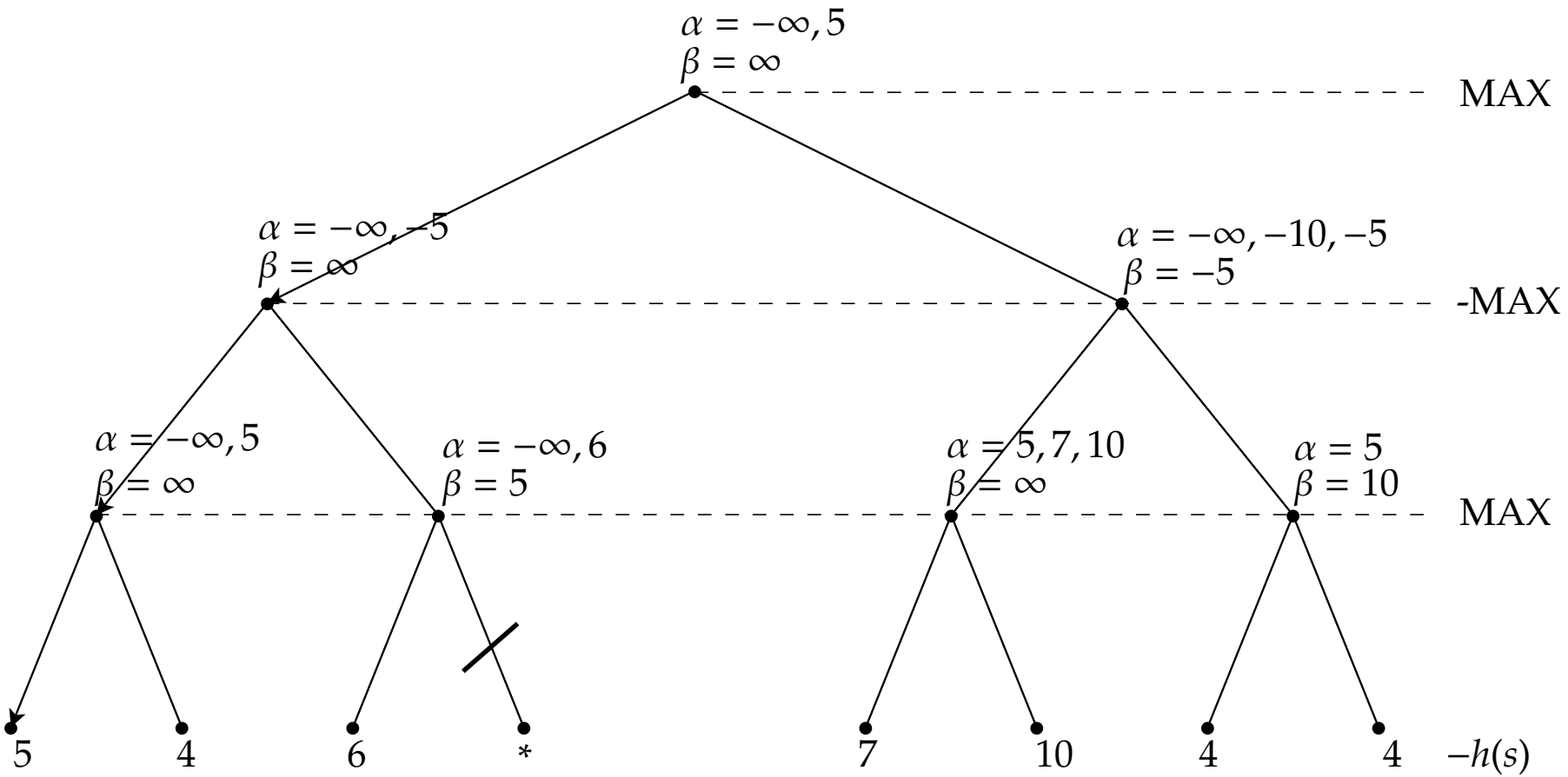
Algorytm *Negamax*

Procedura negaMax(s, d, D, α, β, color)

1. Jeżeli s jest terminalem, to zwróć color $\cdot\, h(s)$.
2. Dla wszystkich stanów t będących potomkami s wykonuj:
   1. $\alpha := \max\{\alpha, -\text{negaMax}(t, d + \frac{1}{2}, D, -\beta, -\alpha, -\text{color})\}$.
   2. Jeżeli $\alpha \geqslant \beta$, to zwróć α.
3. Zwróć α.

Wywołanie zewnętrzne na rzecz maksymalizującego korzenia: negaMax(korzeń, $0, D, -\infty, \infty, 1$).

Ilustracja *Negamax'u* — przykład 1



Algorytm *Negascout*

Procedura negascout(s, d, D, α, β, color)

1. Jeżeli s jest terminalem, to zwróć color $\cdot\, h(s)$.
2. $b := \beta$.
3. Dla wszystkich stanów t będących potomkami s wykonuj:
   1. $v := -\text{negascout}(t, d + \frac{1}{2}, D, -b, -\alpha, -\text{color})$.
   2. Jeżeli t nie jest pierwszym dzieckiem i $D - d > 2 \cdot \frac{1}{2}$ i $b \leqslant v$ (failing high), to:
      1. $v := -\text{negascout}(t, d + \frac{1}{2}, D, -\beta, -v)$. (ponowne przeszukanie szerszym oknem)
   3. $\alpha := \max\{\alpha, v\}$.
   4. Jeżeli $\alpha \geqslant \beta$, to zwróć α. (przycięcie)
   5. $b := \alpha + 1$.
4. Zwróć α.

Warcabnik (Mateusz Bożykowski) (1)

Praca magisterska: **Mateusz Bożykowski, *Implementacja samouczącego się programu do gry w warcaby*, WI, 2009.**

- Warcaby: **międzynarodowe (tzw. polskie)** (100-polowe, gracze mają po 20 pionów), **brazylijskie** (64-polowe, po 12 pionów), **angielskie** (64-polowe, po 12 pionów, damki ruszają się o 1 pole).
- Implementacja *przycinania* α-β z wykorzystaniem tablicy transpozycji i algorytmu *Quiescence*.
- Wiele programów warcabowych grających wg różnych heurystyk konkurowało przeciw sobie w ramach **ewolucji genetycznej**.
- **Osobniki tożsame z heurystykami** oceniającymi pozycję (różne SI). Najprostsza heurystyka **materialna, symetryczna**:

$$h = w_1 P_p + w_2 K_p - w_1 P_o - w_2 K_o, \tag{13}$$

gdzie P, K oznaczają odpowiednio liczbę pionów i damek (pawns, kings) a indeksy p, o oznaczają odpowiednio gracza oceniającego (player) i przeciwnika (opponent). Parametrami do optymalizacji genetycznej są w_1, w_2.

- Parametry kodowane liczbami całkowitymi, losowanymi początkowo z $[-100, 100]$. Liczby nie wyszły poza ten zakres w trakcie ewolucji.

Warcabnik (Mateusz Bożykowski) (2)

- Nie można przyporządkować liczbowego przystosowania osobnikom, a nawet rankingu — zdarzały się przypadki **„kamień-nożyce-papier”** (ang. *three-ways-draw*): A wygrywa z B, B wygrywa z C, C wygrywa z A.
- Nasuwa się **selekcja turniejowa**. Problemy: (1) często spotykane remisy (kogo wówczas przenieść do populacji następnej?), (2) możliwość utraty najlepszego osobnika, (3) wskazanie jednego najlepszego osobnika w końcowej populacji.
- Ostatecznie: **autorski pomysł na selekcję turniejową** z rozmiarem populacji będącym potęgą dwójki $n = 2^m$. Osobniki parowane losowe do meczy, $n/2$ populacji napełniane zwycięzcami turniejów (w przypadku remisu dodawane 1 dziecko skrzyżowanych rodziców), reszta populacji dopełniana iteracyjnie zwycięzcami pojedynków pomiędzy wcześniej dodanymi osobnikami.
- Zwycięzca ostatniego pojedynku (zwycięzca wśród zwycięzców) uznawany za najlepszego osobnika w ostatniej populacji.

Warcabnik (Mateusz Bożykowski) (3)

- **Krzyżowanie liniowe** — losujemy liczbę $\alpha \in (0,1)$, dziecko C rodziców A i B otrzymuje heurystykę:
$$w_i(C) = \alpha w_i(A) + (1-\alpha) w_i(B). \tag{14}$$
- **Krzyżowanie jednorodne** — dla każdego w_i losujemy, od którego rodzica pochodzi (i ustawiamy bez zmian). Sugerowana dodatkowo mutacji.
- **Mutacja** o stałym promieniu, każdy gen zmieniany o wartość losową z $[-20, 20]$. Prawdopodobieństwo mutacji wygaszane liniowo od 0.9 w pierwszej iteracji do 0.3 w ostatniej iteracji.
- Możliwość nastawy różnej głębokości drzewa w zależności od numeru iteracji genetycznej (im AG dalej tym większe drzewo).

Warcabnik (Mateusz Bożykowski) (4)

Badane heurystyki:

- **materialna, niesymetryczna**

$$h = w_1 P_p + w_2 K_p + w_3 P_p + w_4 K_p \tag{15}$$

- **materialno-pozycjna, symetryczna** (ogólnie $h = h_p - h_o$)

$$h_s = w_1 P_s + w_2 K_s + w_3 C_s + w_4 S_s + w_5 OT_s + w_6 PT_s + w_7 KD_s, \tag{16}$$

gdzie: C — liczba bierek w centrum, S — liczba bierek po bokach, OT — liczba pionów na linii przemiany przeciwnika (bronienie przemiany), PT — liczba pionów o jedno pole od własnej przemiany, KD — liczba damek na głównej przekątnej.

- **rozszerzona materialno-pozycjna, symetryczna** (ogólnie $h = h_p - h_o$)

$$h_s = w_1 P_s + \cdots + w_7 KD_s + w_8 M_s + \left\lfloor w_9 \frac{D_s}{P_s + K_s} \right\rfloor + w_{10} KD2_s, \tag{17}$$

gdzie: $M \in 0, 1$ — bonus za ruch, D — liczba zdublowanych bierek (stykających się rogami), $KD2$ — liczba damek na podwójnej przekątnej.

Warcabnik (Mateusz Bożykowski) (5)

Badane heurystyki:

- **materialno-wierszowa** — piony mają różne wartości w różnych wierszach.

$$h_s = \sum_{i=1}^{N-1} w_i P_i + w_N K, \tag{18}$$

gdzie: N — liczba wierszy warcabnicy, P_i — liczba pionów w i-tym wierszu.

- **materialno-strukturalna, symetryczna** — powstała w wyniku obserwacji, że heurystyka wierszowa była istotnie słabsza od rozszerzonej materialno-pozycyjnej (próba podratowania wierszowej), oraz obserwacji, że liczba zdublowanych bierek miała wpływ ujemny:

$$h_s = w_1 P_s + w_2 K_s + w_3 OT_s + w_4 PT_s + w_5 I_s + w_6 F_s \tag{19}$$

gdzie: I — liczba niezbijalnych bierek, F — liczba unieruchomionych bierek.

Warcabnik (Mateusz Bożykowski) (6)

Uzyskane wyniki optymalizacji:

- **materialna, niesymetryczna**:

$$P_p = 5, K_p = 12, P_o = -7, K_o = 10.$$

Komentarz: agresywna gra pionami, zbicie cudzego piona podnosi ocenę pozycji o 2. Ostrożna gra damkami w końcówce, bo własne więcej warte od przeciwnika.

- **materialno-pozycyjna, symetryczna**:

$$P_s = 24, K_s = 65, C_s = 1, S_s = 1, OT_s = -11, PT_s = 27, KD_s = 0.$$

Komentarz: dziwi uznanie pionów broniących linii przemiany za negatywne, oraz damek na przekątnej za obojętne.

Warcabnik (Mateusz Bożykowski) (7)

Uzyskane wyniki optymalizacji:

- **rozszerzona materialno-pozycyjna, symetryczna**:

$$P_s = 5, K_s = 16, C_s = 0, S_s = 0, OT_s = 0, PT_s = 6, KD_s = 0, M_s = 0, D_s = -7, KD2_s = 0.$$

Komentarz: zaskakuje wyzerowanie większości parametrów; istotne wydają się piony tuż przed przemianą i kara za dublowanie bierek.

- **rozszerzona materialno-wierszowa, symetryczna**:

$$P_1 = 2, P_2 = 1, P_3 = 2, P_4 = 2, P_5 = 2, P_6 = 2, P_7 = 1, P_8 = 3, P_9 = 6, K = 12.$$

Komentarz: interesujące wyrównanie wartości w wierszach od 3 do 6, pion w 8-ym wierszu już zaczyna być więcej warty.

- **rozszerzona materialno-strukturalna, symetryczna**:

$$P_s = 13, K_s = 85, OT_s = 0, TT_s = 6, I_s = 1, F_s = -1.$$

Komentarz: widać, że niezbijalność niesie dodatni wpływ, a unieruchomienie ujemny, co kryło się we wcześniej rozważanym parametrze dublowanie D.

Warcabnik (Mateusz Bożykowski) (7)

Porównanie z darmowymi programami warcabowymi:

- **Dam 2.0** — wysoko ceniony program, rozwijany od 1987 r. Brak możliwości nastawy głębokości drzewa — **porównanie niemiarodajne**. Test: najlepsze SI *Warcabnika* z drzewem 7 półruchów vs. kolejne poziomy *Dam 2.0*. *Warcabnik* wygrywa z poziomami Beginner A, B, remisuje z Beginner C, D, zaczyna przegrywać na poziomie powyżej Beginner (tak naprawdę przegrywa końcówki damkowe).
- **Little polish v0.7** — program Piotra Belinga do gry w warcaby brazylijskie. Ograniczono przeciwnika do 1 *s* na ruch. *Warcabnik* zremisował, mimo że przeciwnik analizował czasami nawet pozycje na 18 półruchów wgłąb. Pojedynki z silniejszymi SI (> 1 *s*) *Warcabnik* przegrywał.
- **Windames 3D** — pozwala na ustawienie 9 poziomów gry. *Warcabnik* wygrywał z trzema pierwszymi poziomami i przegrywał z kolejnymi. Ponieważ czasy wykonywania ruchów na poziomie czwartym u obu podobne, można przypuszczać, że podobna głębokość przeszukiwań.

Warcabnik (Mateusz Bożykowski) (8)

Porównanie z darmowymi programami warcabowymi:

- **Warcaby v.1.2** — program Marcina Żółtkowskiego do gry w warcaby polskie i brazylijskie, pozwala nastawić głębokość przeszukiwań (bardziej miarodajne porównanie). Testy na głębokości 6 półruchów (maks. dopuszczalny u przeciwnika). *Warcabnik* wygrał w obie odmiany i był znacznie szybszy (ok. 2 s na ruch, a przeciwnik ok. 30 s). Ustawiając 1 półruch płycej przeszukiwanie *Warcabnika*, uzyskiwał on remis w brazylijskie.
- **Gui Checkers 1.05+** — program Jona Kreuzera do gry w warcaby angielskie. Pozwala nastawić maks. głębokość i czas. Testy na głębokości 10 półruchów — *Warcabnik* przegrywał nawet po ustawieniu mu 1 półruchu więcej. Wniosek: zbadane heurystyki oceniające niewystarczające.

Brydż — problem „double dummy”

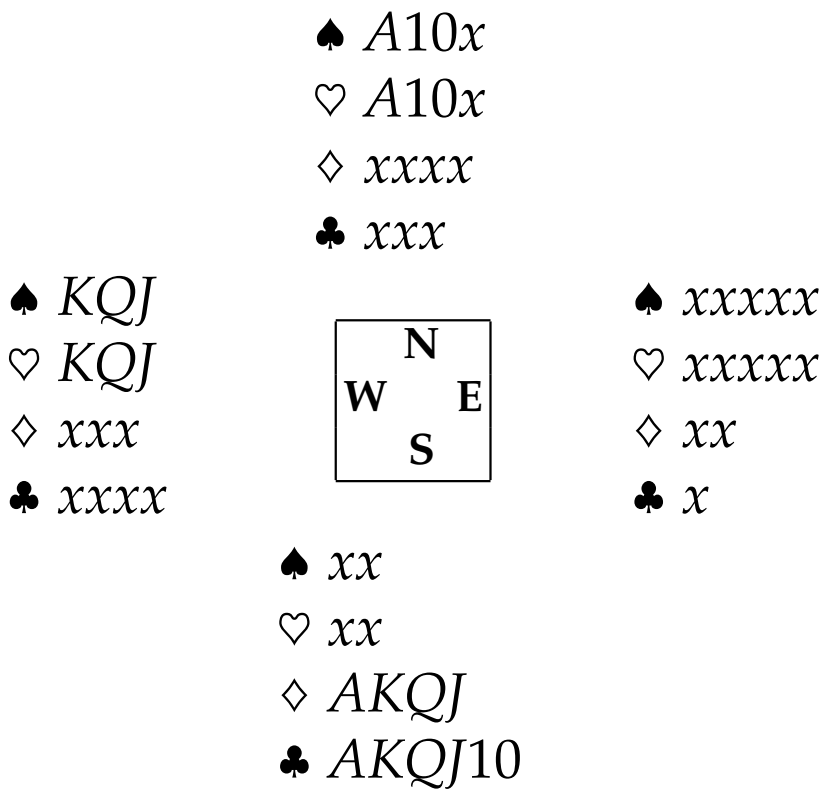
Double dummy

Wersja brydża nieporównawczego jako gry o pełnej informacji. Przydaje się do analizy optymalnej rozgrywki brydżowej, w momencie gdy już całe rozdanie jest wiadome. Istniejące programy: *Deep finesse*, *GIB*.

Praca inżynierska: **Katarzyna Kubasik, *Zastosowanie algorytmów przeszukiwania drzew gier do znajdowania punktów minimaksowych dla problemów “double dummy” w grze w brydża*, WI, 2011.**

- Implementacja *przycinania* α-β z wykorzystaniem tablicy transpozycji.
- Mimo, że 4 graczy N, E, S, W; gracze naprzemienni N i S oraz E i W stanowią pary, które możemy utożsamiać jako dwóch graczy: maksymalizującego i minimalizującego.
- Zagrania każdego z graczy stanowią osobny poziom w drzewie. Pełne drzewo ma $4 \cdot 13 = 52$ poziomy.
- Drzewo przeszukiwań **nie jest naprzemienne poziomami MAX i MIN** — należy sprawdzać, która strona wzięła ostatnią lewę (wziątkę).
- Usprawniające elementy: **bieżące sprawdzanie sekwensów** (np. układ $6, 4, 2$, staje się sekwensem po tym, jeżeli pozostali gracze zagrają 5 i 3); **sortowanie zagrań wg heurystyki: *„skrajne najpierw, środkowe później”***.

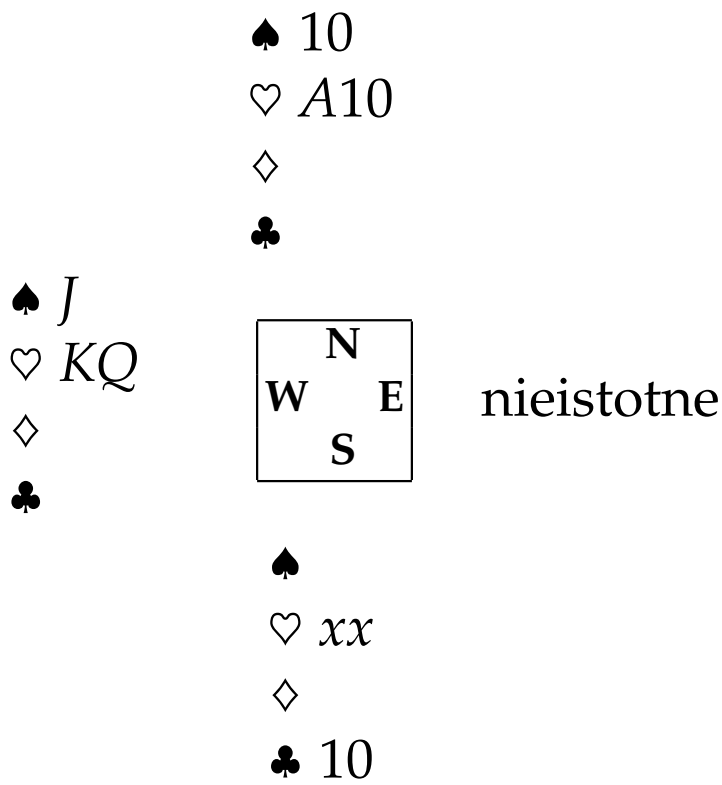
Przykład problemu „double dummy”



Kontrakt: 6 bez atu, grają NS. Pierwsze zagranie W: $K\spadesuit$. Jak ma grać NS, aby wziąć 12 lew?

Rozwiązanie przykładu

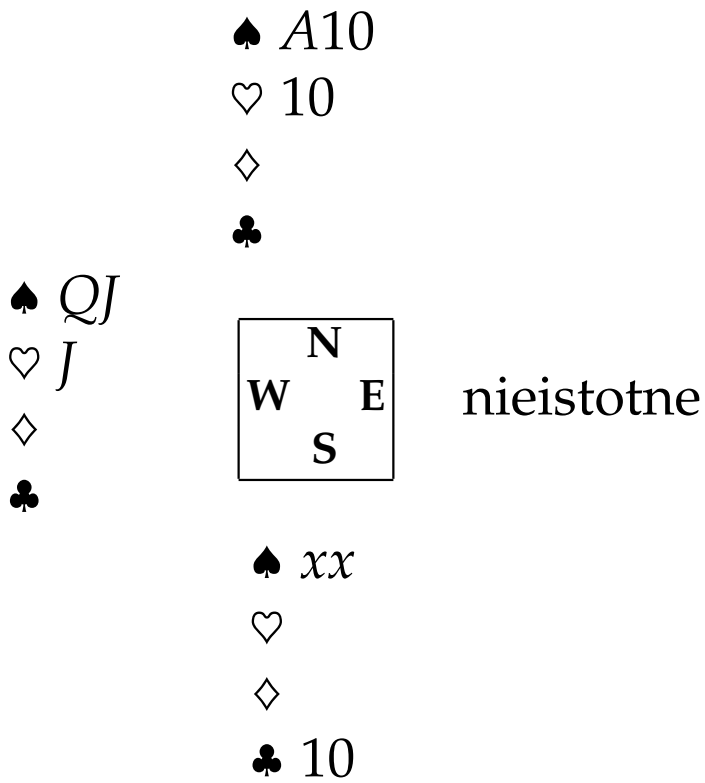
Gracz N powinien **przepuścić** (nie zabijać asem) pierwszą lewę. Po dowolnej kontynuacji W powstanie w końcówce **przymus pikowo-kierowy**. Np. po kontynuacji $Q\spadesuit$.



S gra teraz $10\clubsuit$ i W jest w przymusie. **Bez przepuszczenia w pierwszej lewie przymus nie zaistnieje — pierwsze zagranie N ma skutek 40 poziomów głębiej!**

Rozwiązanie przykładu

Po innej kontynuacji (innym drugim zagraniu W), np. $K\heartsuit$ powstanie końcówka:



nieistotne

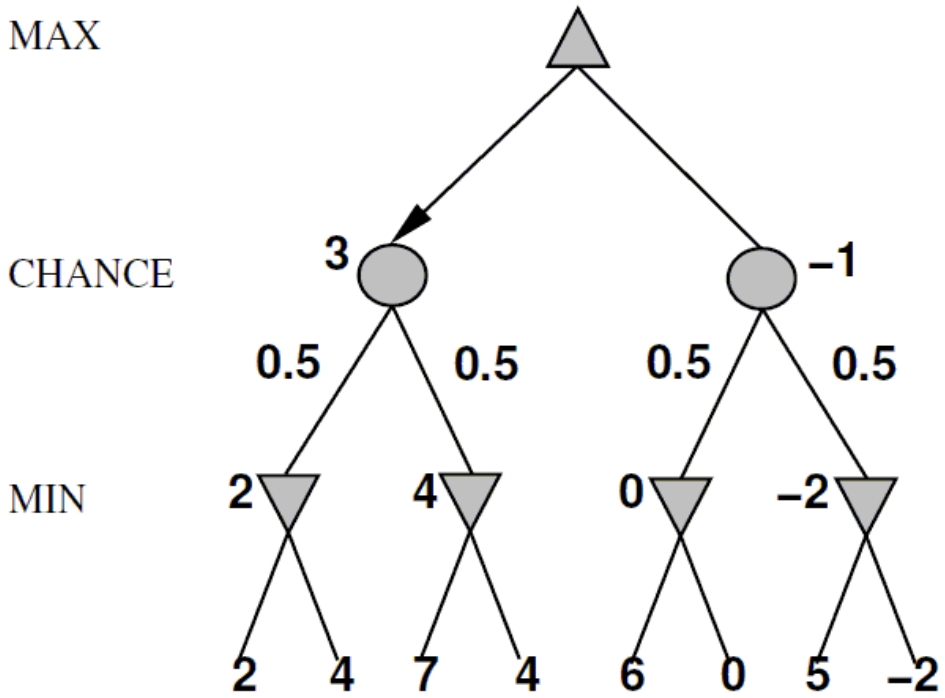
Ponownie S gra 10♣ i W jest w przymusie.

Spis treści

Algorytm *Expectiminimax* (Michie, 1966)

- Przeznaczony do gier, których wynik po części zależy od wyborów graczy, a po części od **czynników losowych**: np. rzut kostką do gry, rzut monetą, zaczerpnięcie karty z zakrytej potasowanej talii, itp.
- Pomysł: oprócz węzłów drzewa realizujących operacje MIN, MAX wprowadza się dodatkowe węzły **CHANCE**, który realizują operację **średniej arytmetycznej** lub **średniej ważonej**.
- Miejsce występowania poziomów CHANCE zależy od reguł gry. W szczególności np. jeżeli każdy gracz przed swoim ruchem rzuca kostką, to mamy przeplot: CHANCE - MAX - CHANCE - MIN - $\cdots$.

Ilustracja *Expectiminimax*

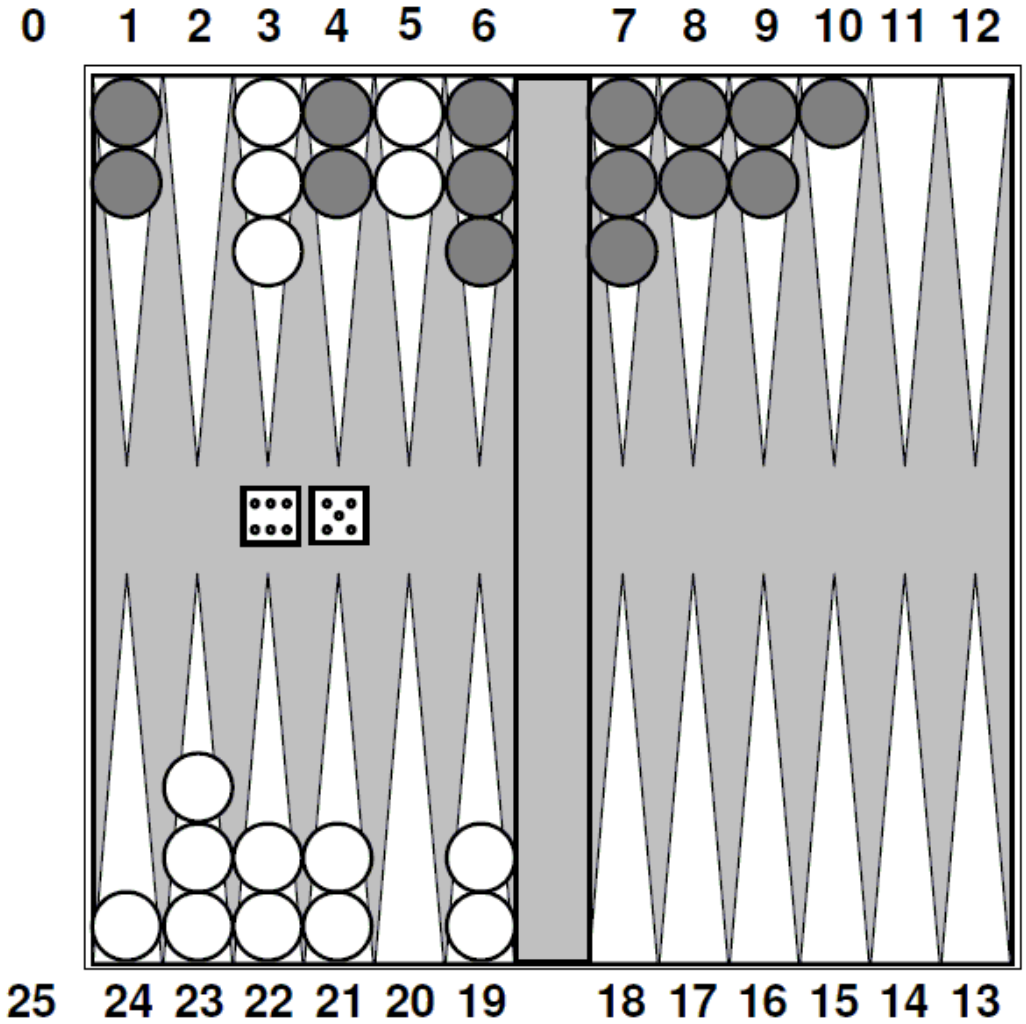


Algorytm *Expectiminimax*

evaluateState(s, d, D) (mieszana procedura rekurencyjna)

1. Oblicz ocenę $h := h(s)$.
2. Jeżeli $h = \pm\infty$ lub $d = D$, to zwróć h.
3. Jeżeli s jest stanem **MIN**:
   1. $v := \infty$.
   2. Dla wszystkich stanów t będących potomkami s:
      $v := \min\{v, \text{evaluateState}(t, d + \frac{1}{2}, D)\}$.
4. Jeżeli s jest stanem **MAX**:
   1. $v := -\infty$.
   2. Dla wszystkich stanów t będących potomkami s:
      $v := \max\{v, \text{evaluateState}(t, d + \frac{1}{2}, D)\}$.
5. Jeżeli s jest stanem **CHANCE**:
   1. $v := 0$.
   2. Dla wszystkich stanów t będących potomkami s:
      $v := v + P(t) \cdot \text{evaluateState}(t, d + \frac{1}{2}, D)$.
6. Zwróć v.

Przykład: *backgammon* (pol. *tryk-trak*)



Przykład: *backgammon* (pol. *tryk-trak*)

- Rzut dwiema kostkami — możliwości: $21 = \binom{2+6-1}{2}$. Rozgałęzianie na poziomie CHANCE: $b = 21$.
- Dla $n = 15$ żetonów, w wyniku „typowego rzutu" (nie „dubla") gracz w maksymalnie swobodnym przypadku może poruszyć wybranym jednym żetonem (suma oczek), lub wybranymi dwoma. Liczba możliwości: $n(n-1) = 210$.
- W przypadku tzw. „dubli" (taka sama liczba oczek na obu kostkach) gracz ma do dyspozycji 4 pojedyncze ruchy o tej samej wartości. Liczba możliwości: $\binom{4+n-1}{4} = 3060$. Duble pojawiają się z prawdopodobieństwem $\frac{1}{6}$.
- Blokady pól zmniejszają istotnie liczbę możliwych ruchów. Średnią wartość rozgałęziania najlepiej oszacować doświadczalnie z gry.
- Gdy głębokość rośnie prawdopodobieństwo danego stanu maleje wykładniczo — dalekie przewidywanie nie jest bardzo wartościowe.
- Program *TDGammon* używa przeglądania na głębokość $D = 2$ (4 półruchy) i bardzo dobrej heurystyki oceniającej pozycję (wystarcza dla gry na poziomie mistrzowskim).

Spis treści

Niepełna informacja (ang. *imperfect information*)

- Np. gry karciane, gdzie nie znamy początkowych kart przeciwnika.
- Teoretycznie, można policzyć prawdopodobieństwo każdego możliwego rozdania/sytuacji — „trochę jak kostka o bardzo wielu ścianach rzucona jeden raz na początku gry”.
- **Pomysł**: obliczyć wypłatę każdego zagrania w każdym rozdaniu i wybrać zagranie o największej wartości oczekiwaną wypłaty.
- Przykład: *GIB* — aktualnie najlepszy program do rozgrywki brydżowej losuje 100 rozdań (próbkowanie Monte-Carlo) zgodnych z dotychczasowymi informacjami i wybiera zagranie, które w średnim przypadku daje największą liczbę lew.
- Szczególny przypadek: **jeżeli istnieje zagranie, które w każdym rozdaniu daje największą wypłatę, to jest ono optymalne**.
- **Uwaga**: powyższa intuicja może być mylna i powodować błędy, jeżeli nierówność jest nieostra (istnieją inne takie zagrania) i jeżeli nie obserwujemy wariancji.
- Tak naprawdę może być koniecznym **wielokrotne uruchamianie próbkowania Monte-Carlo**, przed każdym kolejnym zagraniem (przed każdym przeszukaniem).

Mylna intuicja — przykład 1

- Wylosowane rozdanie 4-lewowe. Rozgrywane „bez atu”. Gracze muszą dokładać do koloru.
- Pewne (znane) karty **gracza A:** $K\spadesuit, Q\spadesuit, A\diamond, A\clubsuit$
- Pewne (znane) karty **gracza B:** $A\spadesuit, J\spadesuit, A\heartsuit$.
- Przypuśćmy, że możliwa **czwarta karta gracza B jest niewiadoma**: $K\diamond$ lub $K\clubsuit$.
- Ocenić zagranie $K\spadesuit$ z punktu widzenia gracza A.

Mylna intuicja — przykład 1

Możliwe rozdanie 1: (gracz A) $K\spadesuit, Q\spadesuit, A\diamond, A\clubsuit : A\spadesuit, J\spadesuit, A\heartsuit, K\diamond$ (gracz B).

Przykładowy ciąg poprawnych zagrań:

1. A: $K\spadesuit$, B: $A\spadesuit$. Pozostałe: (gracz A) $Q\spadesuit, A\diamond, A\clubsuit : J\spadesuit, A\heartsuit, K\diamond$ (gracz B).
2. B: $A\heartsuit$, A: $A\clubsuit$. Pozostałe: (gracz A) $Q\spadesuit, A\diamond : J\spadesuit, K\diamond$ (gracz B).
3. B: $J\spadesuit$, A: $Q\spadesuit$. Pozostałe: (gracz A) $A\diamond : K\diamond$ (gracz B).
4. A: $A\diamond$, B: $K\diamond$

Obie strony biorą po 2 lewy. **Wynik:** 0 **(?)**.

Mylna intuicja — przykład 1

Możliwe rozdanie 2: (gracz A) $K\spadesuit, Q\spadesuit, A\diamondsuit, A\clubsuit : A\spadesuit, J\spadesuit, A\heartsuit, K\clubsuit$ (gracz B).

Przykładowy ciąg poprawnych zagrań:

1. A: $K\spadesuit$, B: $A\spadesuit$. Pozostałe: $Q\spadesuit, A\diamondsuit, A\clubsuit$ (gracz A) : $J\spadesuit, A\heartsuit, K\clubsuit$ (gracz B).
2. B: $A\heartsuit$, A: $A\diamondsuit$. Pozostałe: $Q\spadesuit, A\clubsuit$ (gracz A) : $J\spadesuit, K\clubsuit$ (gracz B).
3. B: $J\spadesuit$, A: $Q\spadesuit$. Pozostałe: $A\clubsuit$ (gracz A) : $K\clubsuit$ (gracz B).
4. A: $A\clubsuit$, B: $K\clubsuit$

Obie strony biorą po 2 lewy. **Wynik:** 0 **(?)**.

Mylna intuicja — przykład 1

Tak naprawdę mamy: (gracz A) $K\spadesuit, Q\spadesuit, A\diamondsuit, A\clubsuit : A\spadesuit, J\spadesuit, A\heartsuit, K*$ (gracz B).

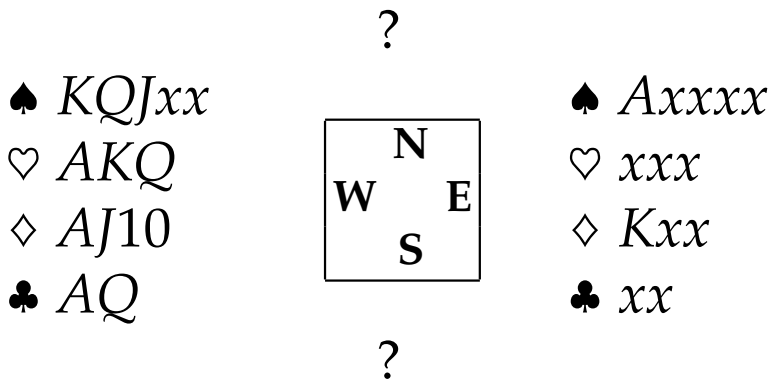
Problem — należy odgadnąć, którego asa wyrzucić w momencie gdy B gra $A\heartsuit$.

1. A: $K\spadesuit$, B: $A\spadesuit$. Pozostałe: $Q\spadesuit, A\diamondsuit, A\clubsuit$ (gracz A) : $J\spadesuit, A\heartsuit, K*$ (gracz B).
2. B: $A\heartsuit$, A: $A\diamondsuit/A\clubsuit$ (?) Pozostałe: $Q\spadesuit, A\clubsuit$ (gracz A) : $J\spadesuit, K*$ (gracz B).

Wynik: $\frac{1}{2} \cdot (-1) + \frac{1}{2} \cdot 0 = -\frac{1}{2}$.

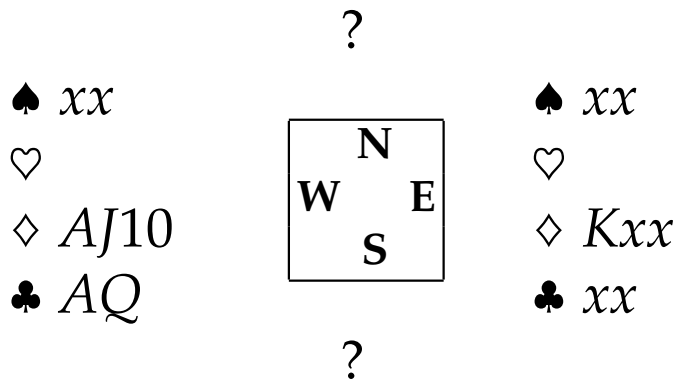
Uwaga: Inne pierwsze zagrania gracza A: $A\diamondsuit$ i $A\clubsuit$ **nie są obarczone dodatnią wariancją i również dają wypłaty** 0.

Mylna intuicja — przykład 2



Kontrakt: 6 pik, grają WE. Pierwsze zagranie N: $x\spadesuit$. Kluczowe brakujące karty: $Q\diamond$, $K\clubsuit$. **Czy istnieje rozgrywka gwarantująca** 12 **lew?**

Mylna intuicja — przykład 2 — rozwiązanie



Optymalna rozgrywka: Zabrać przeciwnikom atuty — piki (grając, jeżeli potrzeba, trzy rundy pików), zgrać trzy rundy kierów. Powstanie powyższa końcówka. Teraz zagrać $A\clubsuit$ i $Q\clubsuit$ poddając $Q\clubsuit$ dobrowolnie! Niezależnie od kontynuacji N lub S, 12 lew jest pewne.

Inna rozgrywka polegająca na próbie złapania $K\clubsuit$ u S ($\approx 50\%$ szans), i złapania $Q\diamondsuit$ u S lub u N ($\approx 50\%$ szans) daje oczekiwaną liczbę lew: $\approx \frac{1}{4}11 + \frac{2}{4}12 + \frac{1}{4}13 = 12$, ale z wariancją $\approx \frac{1}{4}(12-11)^2 + \frac{2}{4}(12-12)^2 + \frac{1}{4}(12-13)^2 = \frac{1}{2}$ lewy.

Wybrane źródła

1. J. von Neuman i O. Morgenstern, *Theory of Games and Economic Behavior*, 1944 (patrz: `http://press.princeton.edu/titles/7802.html`).
2. Strona projektu *Chinook*: `http://webdocs.cs.ualberta.ca/~chinook`.
3. D.E. Knuth, R.W. Moore, *An Analysis of Alpha-Beta Pruning*, Artificial Intelligence, 1975 (patrz: `http://www.eecis.udel.edu/~ypeng/articles/An Analysis of Alpha-Beta Pruning.pdf`).
4. A. Reinefeld, *„An Improvement to the Scout Tree Search Algorithm"*, ICCA Journal, 1983 (patrz: `http://www.top-5000.nl/ps/An improvement to the scout tree search algorithm.pdf`)
5. D. Larameé, *Chess Programming I-V*, 2000. `http://www.gamedev.net/page/resources/_/reference/programming/artificial-intelligence/gaming/chess-programming-part-i-getting-started-r1014`)
6. M. Bożykowski, *Implementacja samouczącego się programu do gry w warcaby*, praca magisterska na WI ZUT, 2009.
7. K. Kubasik, *Zastosowanie algorytmów przeszukiwania drzew gier do znajdowania punktów minimaksowych dla problemów "double dummy" w grze w brydża*, praca inżynierska na WI ZUT, 2011.
8. P. Beling, *Praktyczne aspekty programowania gier logicznych*, praca magisterska, Politechnika Łódzka, 2006.
9. *Expectiminimax tree*, Wikipedia, (patrz: `http://en.wikipedia.org/wiki/Expectiminimax_tree`).
10. Materiały o programie brydżowym *GIB*, (patrz: `http://www.greatbridgelinks.com/gblSOFT/Reviews/SoftwareReview090301.html` oraz `http://www.gibware.com/`).