Kybernetika a umělá inteligence
8. Hraní dvouhráčových her, adversariální prohledávání stavového prostoru

Ing. Michal Pěchouček, Ph.D.
Katedra kybernetiky
ČVUT v Praze, FEL

OPERAČNÍ PROGRAM PRAHA ADAPTABILITA

PRAHA PRAGUE PRAGA PRAG

EVROPSKÁ UNIE

Evropský sociální fond
Praha & EU: Investujeme do vaší budoucnosti

Hraní dvouhráčových her, adversariální prohledávání stavového prostoru

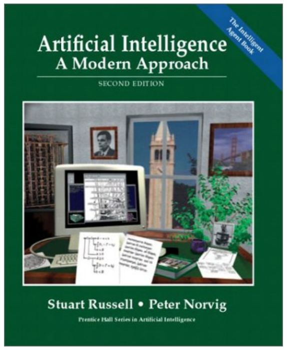
Michal Pěchouček

Department of Cybernetics
Czech Technical University in Prague

http://labe.felk.cvut.cz/~pechouc/kui/games.pdf

Použitá literatura pro umělou inteligenci

:: Artificial Intelligence: A Modern Approach (Second Edition) by Stuart Russell and Peter Norvig, 2002 Prentice Hall.



http://aima.cs.berkeley.edu/

Prohledávání při hraní dvouhráčových her

adversariální prohledávání (adversarial search) stavového prostoru (hraní her), implementuje inteligencí rozhodování v komutativním prostředí, kde dva nebo více agentů mají konfliktní cíle.

teorie her – komplikovaná vědní disciplína, součást ekonomiky která analyzuje chování jednotlivých hráčů a výhodnost jejich strategií (především s ohledem na stabilitu, maximalizaci společného zisku, atp.) ve vícehráčovém prostředí.

Prohledávání při hraní dvouhráčových her

adversariální prohledávání (adversarial search) stavového prostoru (hraní her), implementuje inteligencí rozhodování v komutativním prostředí, kde dva nebo více agentů mají konfliktní cíle.

teorie her – komplikovaná vědní disciplína, součást ekonomiky která analyzuje chování jednotlivých hráčů a výhodnost jejich strategií (především s ohledem na stabilitu, maximalizaci společného zisku, atp.) ve vícehráčovém prostředí.

V umělé inteligence budeme především pracovat s následujícím typem her:

	deterministické	s prvkem náhody
s úplnou informací	šachy, go, reversi	backgamon
s neúplnou informací	stratego, wargaming	bridge, poker, scrabble

Prohledávání při hraní dvouhráčových her

adversariální prohledávání (adversarial search) stavového prostoru (hraní her), implementuje inteligencí rozhodování v komutativním prostředí, kde dva nebo více agentů mají konfliktní cíle.

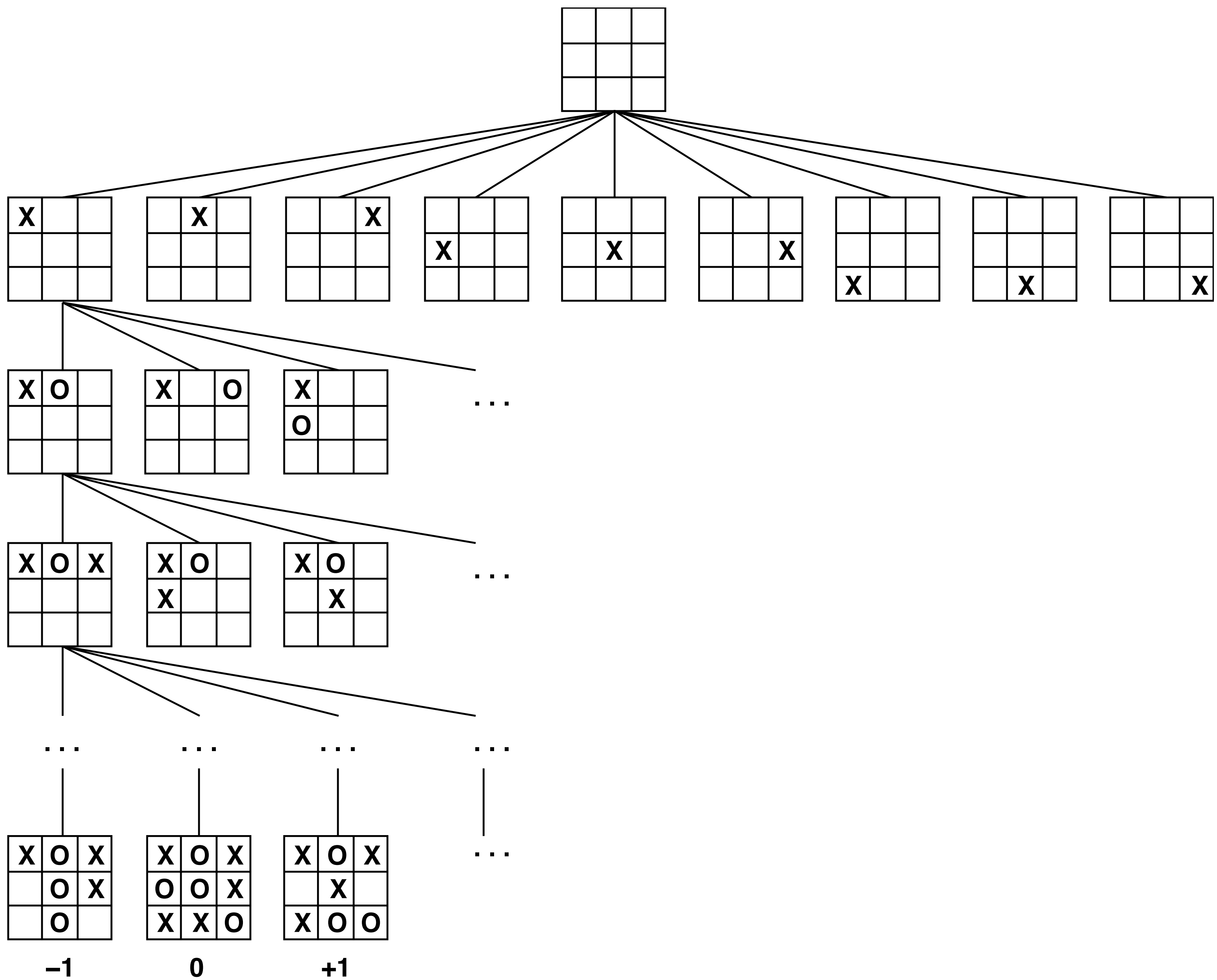
teorie her – komplikovaná vědní disciplína, součást ekonomiky která analyzuje chování jednotlivých hráčů a výhodnost jejich strategií (především s ohledem na stabilitu, maximalizaci společného zisku, atp.) ve vícehráčovém prostředí.

V umělé inteligence budeme především pracovat s následujícím typem her:

	deterministické	s prvkem náhody
s úplnou informací	šachy, go, reversi	backgamon
s neúplnou informací	stratego, wargaming	bridge, poker, scrabble

Stavový prostor hry (**herní strom**) je dán opět počátečním stavem, stavovým operátorem, testem na ukončení hry a užitkovou funkcí.

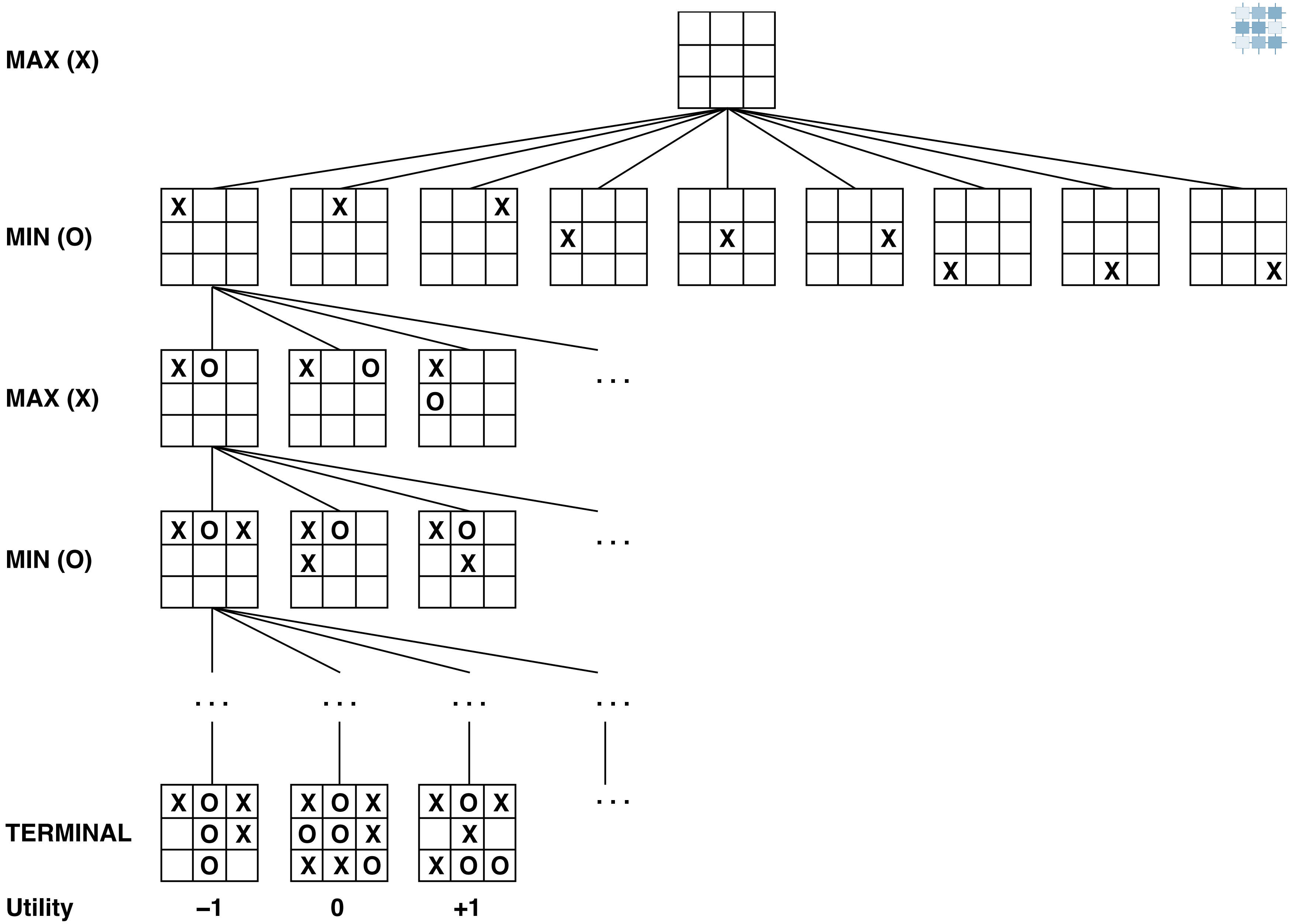
Cílem adversariálního prohledávání je nalézt nikoliv stav prostoru, nýbrž herní strategii. Strategie vybere nejvhodnější tah s ohledem na racionalitu protihráče. **Optimální herní strategie** je taková strategie (nepřesně), pro kterou neexistuje žádná lepší strategie, která by vedla k lepšímu výsledku při hře s bezchybným oponentem.


–1
0
+1

Minimax Algoritmus

MINIMAX algoritmus dělí stavový prostor do MAX a MIN úrovní. Na každé MAX úrovni hráč A vybere tah s maximálním užitkem a na každé MIN úrovni vybere protihráč tah naopak minimalizující užitek hráče A.

http://ai-depot.com/LogicGames/MiniMax.html


MAX (X)
MIN (O)
MAX (X)
MIN (O)
TERMINAL
Utility
–1
0
+1

Minimax Algoritmus

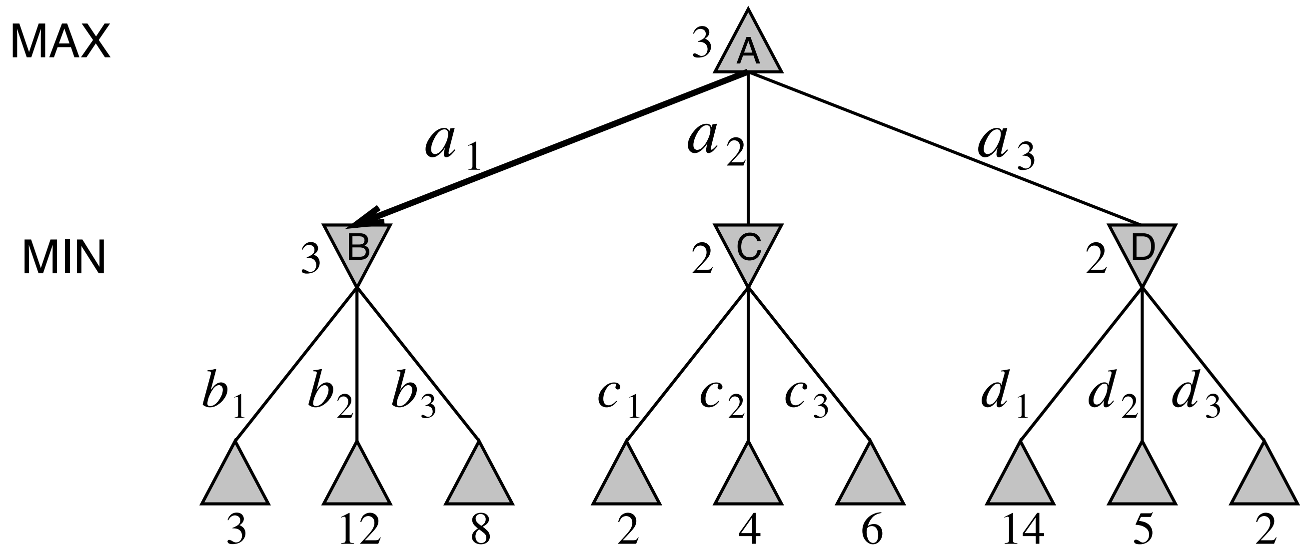
MINIMAX algoritmus dělí stavový prostor do MAX a MIN úrovní. Na každé MAX úrovni hráč A vybere tah s maximálním užitkem a na každé MIN úrovni vybere protihráč tah naopak minimalizující užitek hráče A.

http://ai-depot.com/LogicGames/MiniMax.html

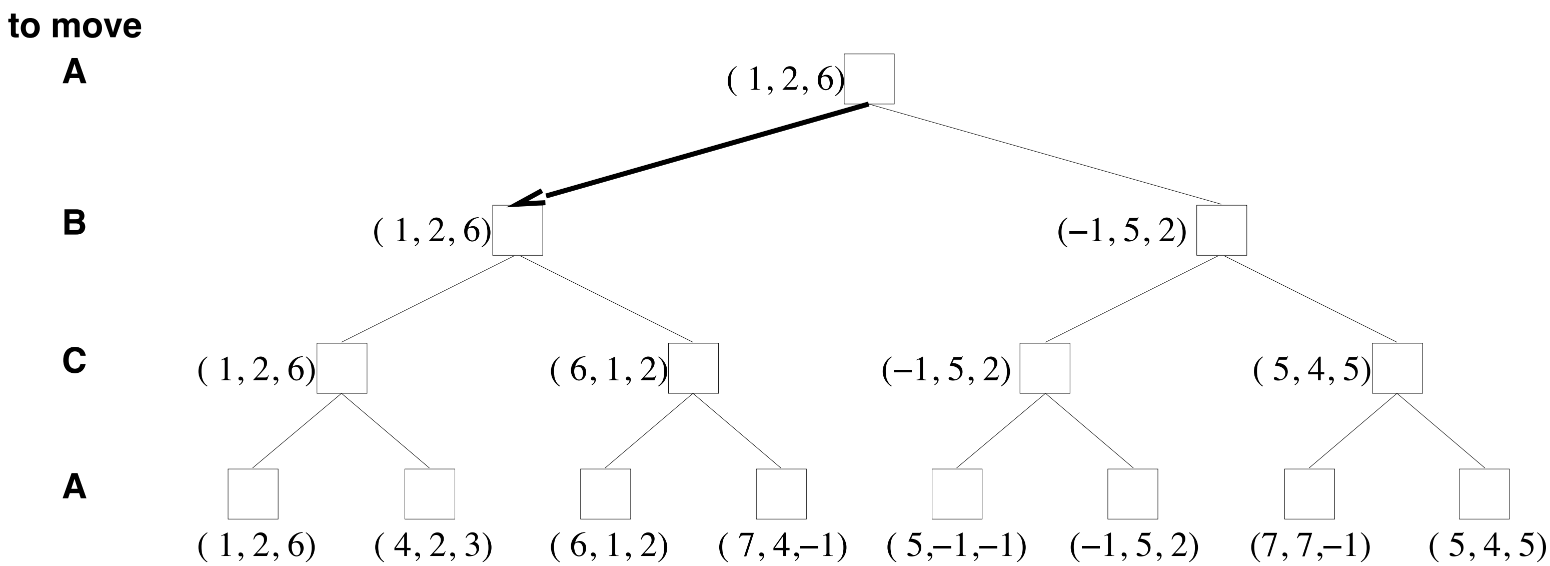
Každý uzel ohodnotíme tzv. MINIMAX hodnotou:

$$\texttt{minimax}(n) = \begin{cases} \texttt{utility}(n) & \text{pro } n \text{ terminalni uzel} \\ \max_{s \in successors(n)} \texttt{minimax}(n) & \text{pro } n \text{ je MAX uzel} \\ \min_{s \in successors(n)} \texttt{minimax}(n) & \text{pro } n \text{ je MIN uzel} \end{cases}$$

Minimax Algoritmus



Minimax Algoritmus



Minimax algoritmus

function MINIMAX-DECISION(*state*) **returns** *an action*
  inputs: *state*, current state in game

  return the a in ACTIONS(*state*) maximizing MIN-VALUE(RESULT(a, *state*))

---

function MAX-VALUE(*state*) **returns** *a utility value*
  if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
  $v \leftarrow -\infty$
  for a, s in SUCCESSORS(*state*) **do** $v \leftarrow$ MAX(v, MIN-VALUE(s))
  return v

---

function MIN-VALUE(*state*) **returns** *a utility value*
  if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
  $v \leftarrow \infty$
  for a, s in SUCCESSORS(*state*) **do** $v \leftarrow$ MIN(v, MAX-VALUE(s))
  return v

Vlastnosti Algoritmu MinMax

- **úplné**: ANO (je-li prostor konečné)
- **čas**:

Vlastnosti Algoritmu MinMax

- **úplné**: ANO (je-li prostor konečné)
- **čas**: $O(b^d)$
- **paměť**:

Vlastnosti Algoritmu MinMax

- **úplné**: ANO (je-li prostor konečné)
- **čas**: $O(b^d)$
- **paměť**: $O(bd)$
- **optimální**:

Vlastnosti Algoritmu MinMax

- **úplné**: ANO (je-li prostor konečné)
- **čas**: $O(b^d)$
- **paměť**: $O(bd)$
- **optimální**: ano

Cut-off search

Problém minimaxu je, že počet stavů hry roste exponenciálně s počtem tahů. V reálných hrách je prostor hry ohromný a nelze ho celý prohledat v rozumném čase. Tento problém lze řešit pomocí:

- omezením hloubky d – `terminal_test` nahradíme `cut_off_test`
- odhadu místo přesné hodnoty užitku v případě, že $d < b$ – `utility` nahradíme `eval` .

příklad funkce `eval` může být:

- počet vyřazených figurek
- vážený součet počtu vyřazených figurek
- vážený součet vhodnosti strategickém umístěni každé figurky

Cut-off search

Problém minimaxu je, že počet stavů hry roste exponenciálně s počtem tahů. V reálných hrách je prostor hry ohromný a nelze ho celý prohledat v rozumném čase. Tento problém lze řešit pomocí:

- omezením hloubky d – `terminal_test` nahradíme `cut_off_test`
- odhadu místo přesné hodnoty užitku v případě, že $d < b$ – `utility` nahradíme `eval` .

příklad funkce `eval` může být:

- počet vyřazených figurek
- vážený součet počtu vyřazených figurek
- vážený součet vhodnosti strategickém umístěni každé figurky

$$\texttt{eval}(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s) = \sum_{n}^{i=1} w_i f_i(s)$$

např., pro $w_1 = 9$, $f_1(s) =$ `(number_of_white_queens) - (number_of_black_queens)`

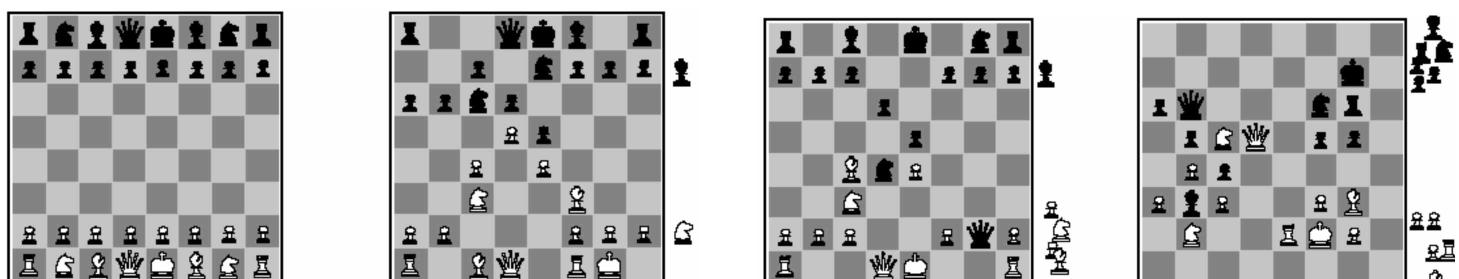
Cut-off search

Problém minimaxu je, že počet stavů hry roste exponenciálně s počtem tahů. V reálných hrách je prostor hry ohromný a nelze ho celý prohledat v rozumném čase. Tento problém lze řešit pomocí:

- omezením hloubky d – `terminal_test` nahradíme `cut_off_test`
- odhadu místo přesné hodnoty užitku v případě, že $d < b$ – `utility` nahradíme `eval` .

příklad funkce `eval` může být:

- počet vyřazených figurek
- vážený součet počtu vyřazených figurek
- vážený součet vhodnosti strategickém umístěni každé figurky



(a) White to move
Fairly even

(b) Black to move
White slightly better

(c) White to move
Black winning

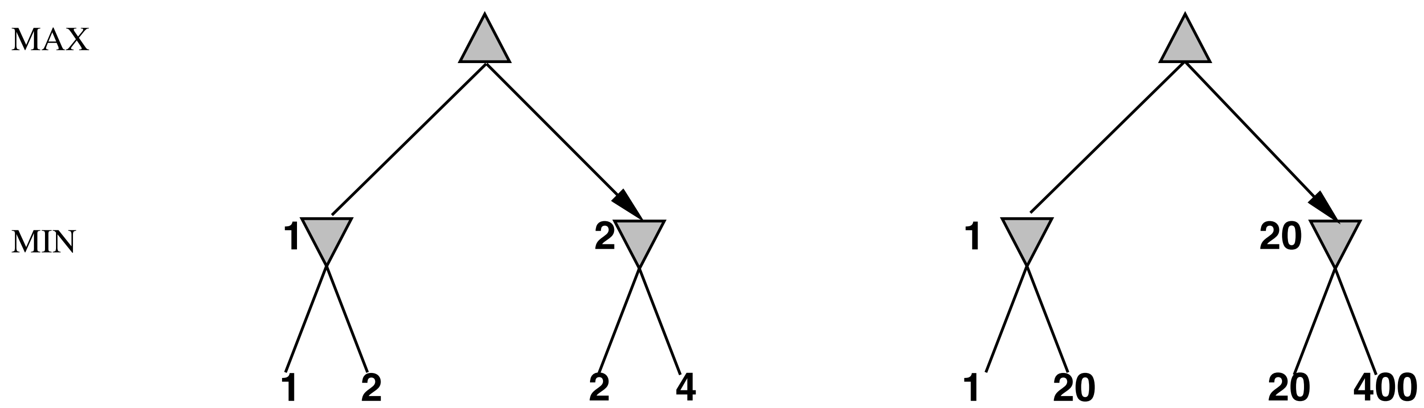
(d) Black to move
White about to lose

Cut-off search

degrese: přesné hodnoty `eval` nejsou důležité - korektní chování algoritmu je zachováno při libovolné monotónní transformaci

Cut-off search

degrese: přesné hodnoty `eval` nejsou důležité - korektní chování algoritmu je zachováno při libovolné monotónní transformaci



Cut-off search

degrese: přesné hodnoty `eval` nejsou důležité - korektní chování algoritmu je zachováno při libovolné monotónní transformaci

problémy, zlepšení:

Cut-off search

degrese: přesné hodnoty `eval` nejsou důležité - korektní chování algoritmu je zachováno při libovolné monotónní transformaci

problémy, zlepšení:

- je třeba ohodnocovat pomocí `eval` jen **klidné** (quiescent) stavy – stavy, které nezpůsobí následnou výraznou změnu v hodnotě

Cut-off search

degrese: přesné hodnoty `eval` nejsou důležité - korektní chování algoritmu je zachováno při libovolné monotónní transformaci

problémy, zlepšení:

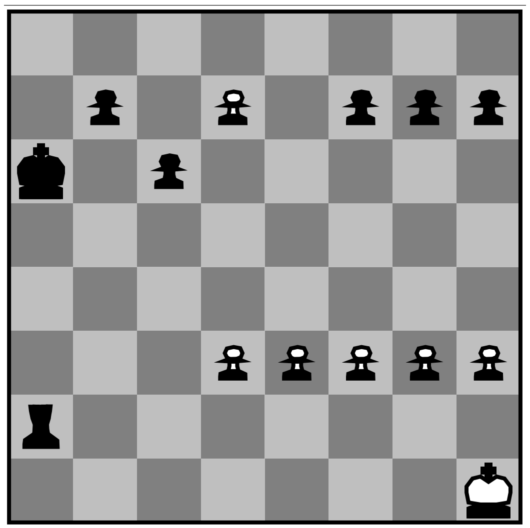
- je třeba ohodnocovat pomocí `eval` jen **klidné** (quiescent) stavy – stavy, které nezpůsobí následnou výraznou změnu v hodnotě
- je třeba zabránit **horizontálnímu efektu** – situaci, kdy program musí udělat tah, který způsobí velkou ztrátu

Cut-off search

degrese: přesné hodnoty `eval` nejsou důležité - korektní chování algoritmu je zachováno při libovolné monotónní transformaci

problémy, zlepšení:

- je třeba ohodnocovat pomocí `eval` jen **klidné** (quiescent) stavy – stavy, které nezpůsobí následnou výraznou změnu v hodnotě
- je třeba zabránit **horizontálnímu efektu** – situaci, kdy program musí udělat tah, který způsobí velkou ztrátu



Cut-off search

degrese: přesné hodnoty `eval` nejsou důležité - korektní chování algoritmu je zachováno při libovolné monotónní transformaci

problémy, zlepšení:

- je třeba ohodnocovat pomocí `eval` jen **klidné** (quiescent) stavy – stavy, které nezpůsobí následnou výraznou změnu v hodnotě
- je třeba zabránit **horizontálnímu efektu** – situaci, kdy program musí udělat tah, který způsobí velkou ztrátu
- lze použít **singulární extenzi** – tah, který jde za `cut-off`, ale jasně zlepšuje `eval` hodnotu (podobné jako quiescent prohledávání ale s $b = 1$)

Cut-off search

degrese: přesné hodnoty `eval` nejsou důležité - korektní chování algoritmu je zachováno při libovolné monotónní transformaci

problémy, zlepšení:

- je třeba ohodnocovat pomocí `eval` jen **klidné** (quiescent) stavy – stavy, které nezpůsobí následnou výraznou změnu v hodnotě
- je třeba zabránit **horizontálnímu efektu** – situaci, kdy program musí udělat tah, který způsobí velkou ztrátu
- lze použít **singulární extenzi** – tah, který jde za `cut-off`, ale jasně zlepšuje `eval` hodnotu (podobné jako quiescent prohledávání ale s $b = 1$)

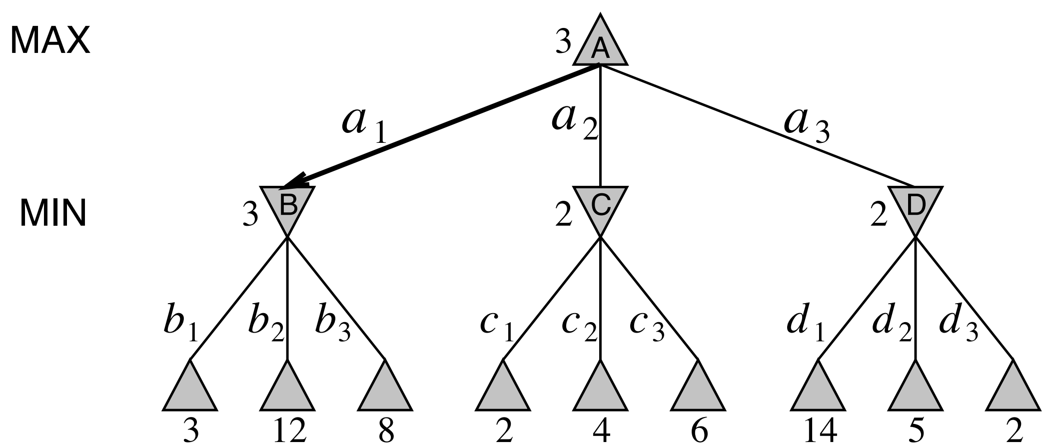
Mějme k dispozici 3 minuty s a uvažujme 10^6 operací za sekundu. Můžeme tedy prohledat $50 * 10^6$ uzlů na tah což je $\approx 35^5$. V šachách můžeme tedy pracovat s hloubkou 5.

Alfa-Beta Prořezávání

Velikost stavového prostoru hry lze rovněž efektivně zmenšit pomocí metody **alfa-beta prořezávání**. Tato metoda umožní identifikovat části stavového prostoru, které jsou nalezení optimálního řešení neelegantně. Při aplikaci na standardní stavový prostor vrátí stejnou strategii jako Minimax a prořeže nerelevantní části prostoru.

Alfa-Beta Prořezávání

Velikost stavového prostoru hry lze rovněž efektivně zmenšit pomocí metody **alfa-beta prořezávání**. Tato metoda umožní identifikovat části stavového prostoru, které jsou nalezení optimálního řešení neelegantně. Při aplikaci na standardní stavový prostor vrátí stejnou strategii jako Minimax a prořeže nerelevantní části prostoru.



$$
\begin{aligned}
\texttt{min-max}(A) &= \max(\min(3,12,8),\min(2,4,6),\min(14,5,2)) \\
&= \max(3,\min(2,x,y),2) \\
&= \max(3,z,2), z \le 2
\end{aligned}
$$

Alfa-Beta Prořezávání

Velikost stavového prostoru hry lze rovněž efektivně zmenšit pomocí metody **alfa-beta prořezávání**. Tato metoda umožní identifikovat části stavového prostoru, které jsou nalezení optimálního řešení neelegantně. Při aplikaci na standardní stavový prostor vrátí stejnou strategii jako Minimax a prořeže nerelevantní části prostoru.

Klasický Minimax algoritmus rozšíříme následujícím způsobem:

- zavedeme hodnotu:
  - α nejlepší známá hodnota pro uzel MAX
  - β nejlepší známá hodnota pro uzel MIN
- na každé MAX úrovni před tím než ohodnotíme následníky, rovnáme `minmax` hodnotu s hodnotou β. je-li `minmax` $> \beta$ pak se tato část stromu se neprohledává
- na každé MIN úrovni před tím než ohodnotíme následníky, rovnáme `minmax` hodnotu s hodnotou α. je-li `minmax` $< \alpha$ pak se tato část stromu se neprohledává

function ALPHA-BETA-DECISION($state$) **returns** an action
  return the a in ACTIONS($state$) maximizing MIN-VALUE(RESULT($a, state$))

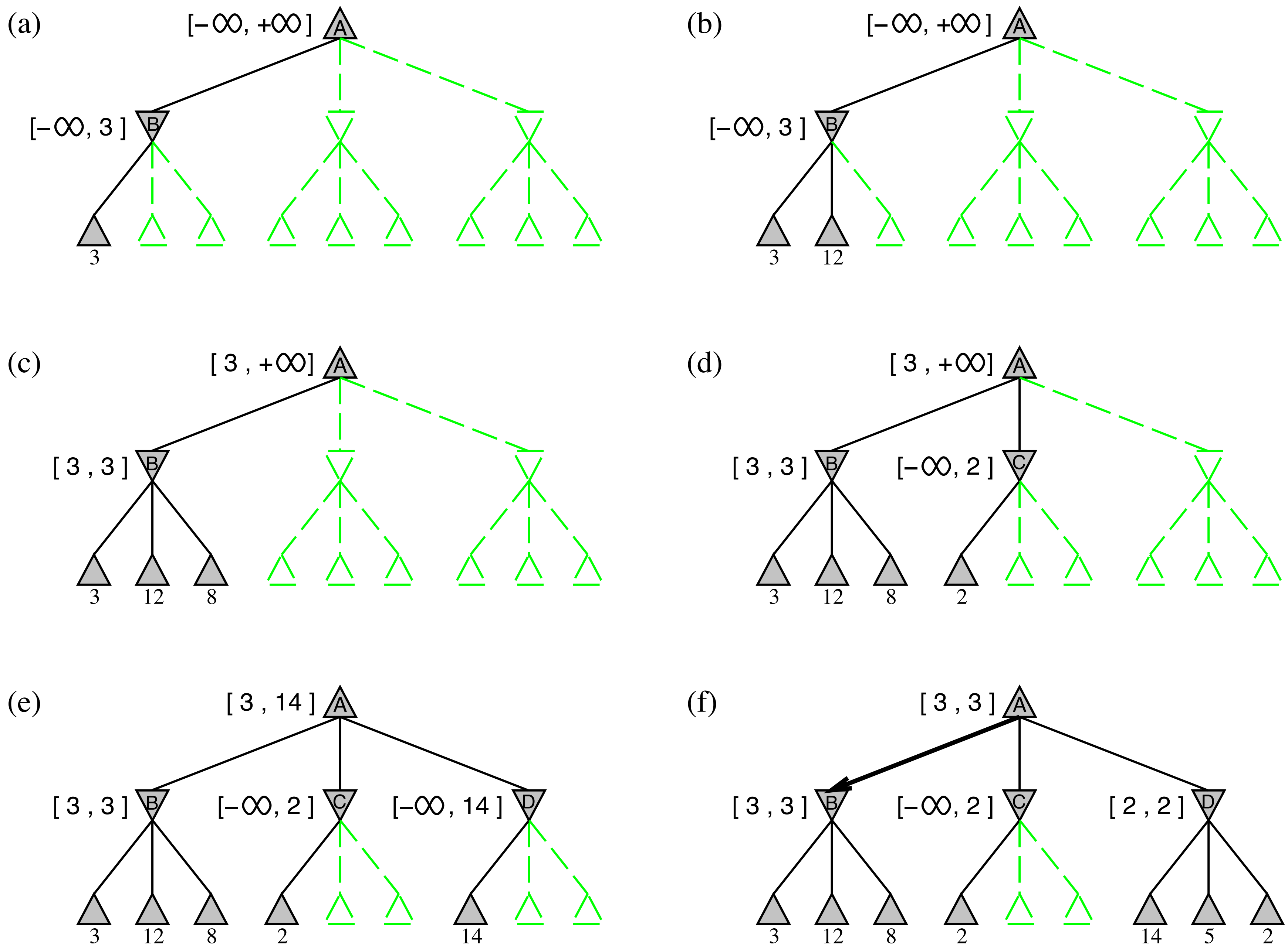
---

function MAX-VALUE($state, \alpha, \beta$) **returns** *a utility value*
  inputs: $state$, current state in game
    α, the value of the best alternative for MAX along the path to $state$
    β, the value of the best alternative for MIN along the path to $state$

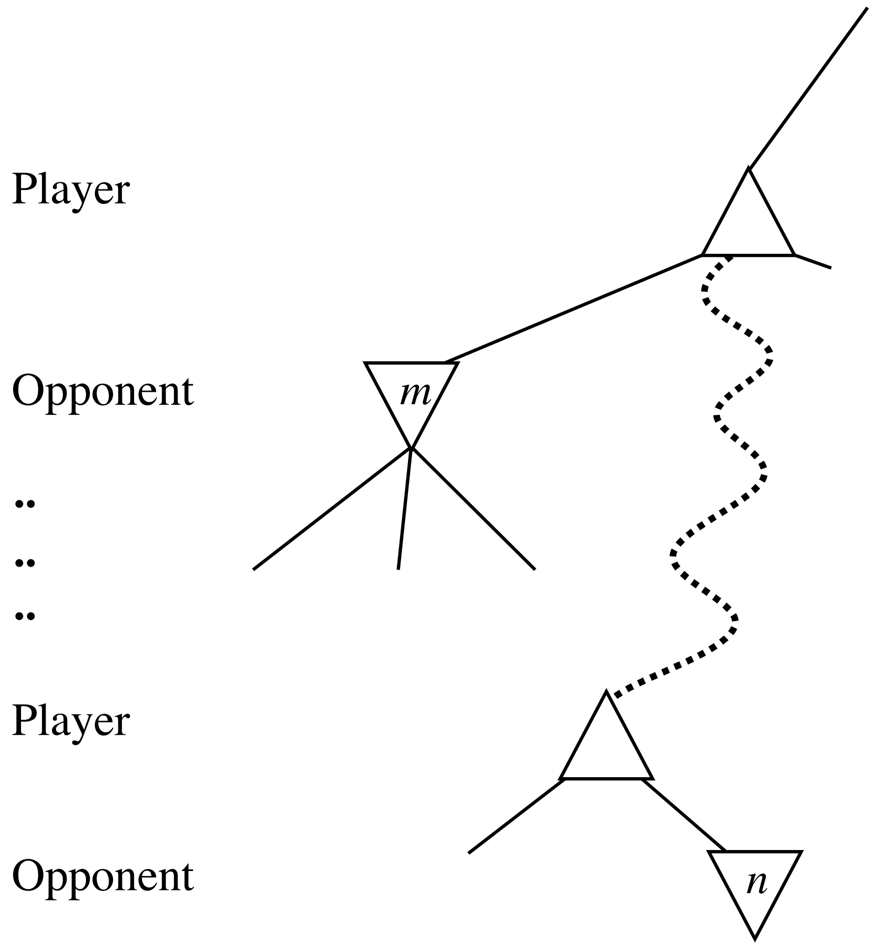
  if TERMINAL-TEST($state$) **then return** UTILITY($state$)
  $v \leftarrow -\infty$
  for $a,\ s$ in SUCCESSORS($state$) **do**
    $v \leftarrow$ MAX(v, MIN-VALUE(s, α, β))
    if $v \geq \beta$ **then return** v
    $\alpha \leftarrow$ MAX(α, v)
  return v

---

function MIN-VALUE($state, \alpha, \beta$) **returns** *a utility value*
  same as MAX-VALUE but with roles of α, β reversed


(a)
[−∞, +∞]
A
[−∞, 3]
B
3
(b)
[−∞, +∞]
A
[−∞, 3]
B
3
12
(c)
[3 , +∞]
A
[3 , 3]
B
3
12
8
(d)
[3 , +∞]
A
[3 , 3]
B
[−∞, 2]
C
3
12
8
2
(e)
[3 , 14]
A
[3 , 3]
B
[−∞, 2]
C
[−∞, 14]
D
3
12
8
2
14
(f)
[3 , 3]
A
[3 , 3]
B
[−∞, 2]
C
[2 , 2]
D
3
12
8
2
14
5
2

Alfa-Beta Prořezávání



Vlastnosti Alpha-Beta Prořezávání

- prořezávání nemá vliv na výsledek

Vlastnosti Alpha-Beta Prořezávání

- prořezávání nemá vliv na výsledek
- lze dokázat, že časová náročnost klesne na $O(b^{d/2})$ v případě, že vždy vybere nejlepší expandand (to implikuje možnost zdvojnásobit hloubku prohledávání)

Vlastnosti Alpha-Beta Prořezávání

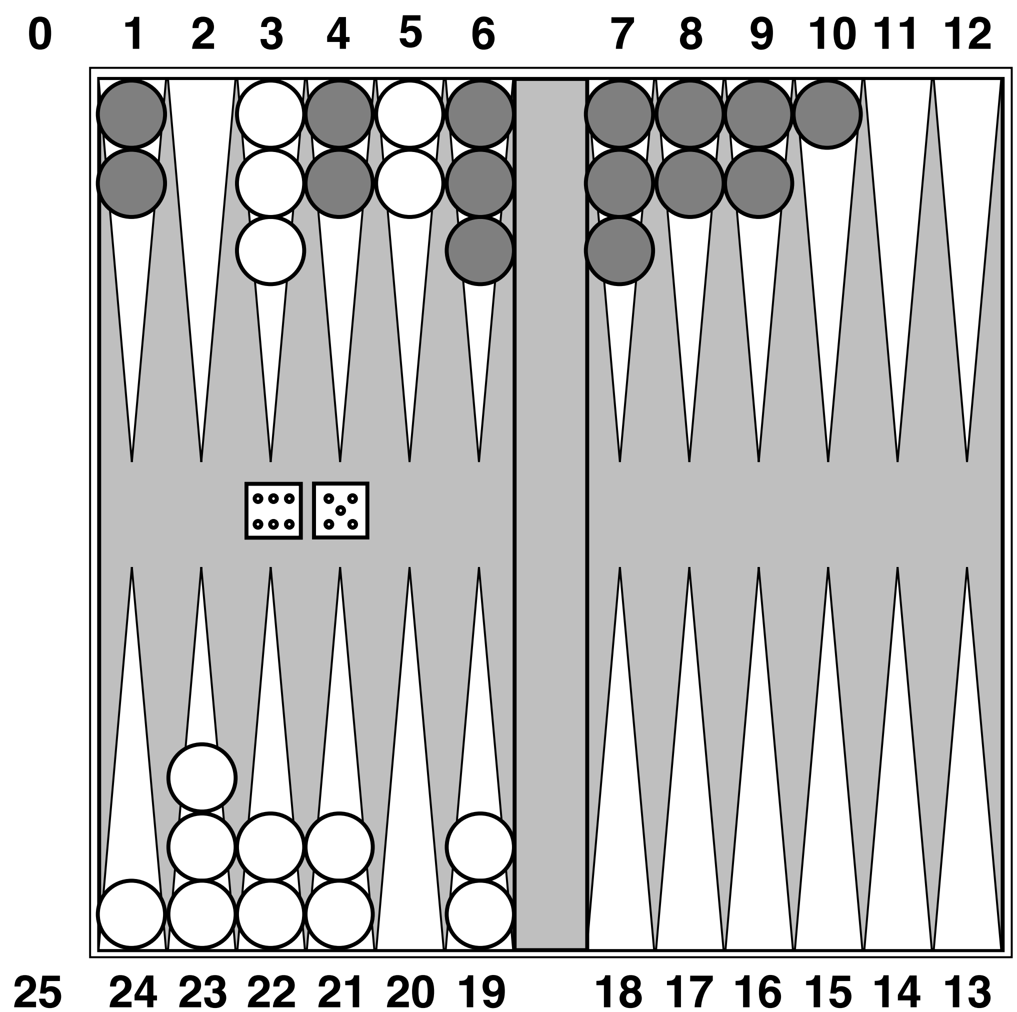
- prořezávání nemá vliv na výsledek
- lze dokázat, že časová náročnost klesne na $O(b^{d/2})$ v případě, že vždy vybere nejlepší expandand (to implikuje možnost zdvojnásobit hloubku prohledávání)
- při náhodném výběru časová náročnost klesne na $O(b^{3d/4})$

Úspěšné herní algoritmy

- **Dáma:** v roce 1994 porazil poprvé po 40 letech algoritmus chinook mistryni světa v dámě Marion Tinsley. Bylo použito databáze koncových tahů pro všechny pozice, které zahrnovali 8 a méně kamenů na šachovnici. Koncových tahů bylo celkem 443,748,401,247
- **Šachy:** v roce 1997 porazil Deep Blue Garyho Kasparova v šestikolovém zápasu. Deep Blue prohledal 200 million pozic za vteřinu a použil velmi sofistikované (a tajné) metody evaluace, které místy umožnily prohledávání do hloubky 40 tahů
- **Othello:** lidé odmítají hrát s počítačem ...
- **Go:** lidé odmítají hrát s počítačem ... (je přespříliš dobrý).
  V Go je $b > 300$. Většina algoritmu používá databáze tahů.

Hry s prvkem náhody

Aplikace klasické metody MINIMAXU, kdy MINIMAX hodnoty jsou nahrazeny očekávanými hodnotami – EMINMAX:


0 1 2 3 4 5 6 7 8 9 10 11 12
25 24 23 22 21 20 19 18 17 16 15 14 13

Hry s prvkem náhody

Aplikace klasické metody MINIMAXU, kdy MINIMAX hodnoty jsou nahrazeny očekávanými hodnotami – EMINMAX:

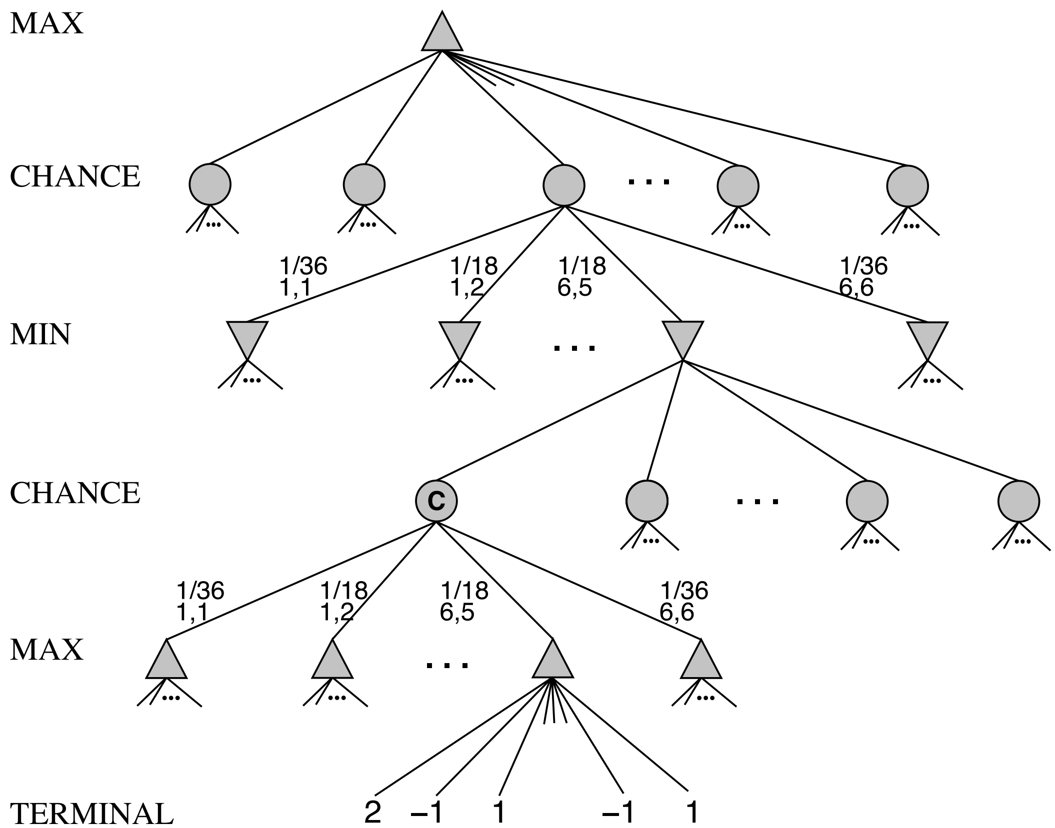
Hry s prvkem náhody

Aplikace klasické metody MINIMAXU, kdy MINIMAX hodnoty jsou nahrazeny očekávanými hodnotami – EMINMAX:

$$\texttt{eminimax}(n) = \begin{cases} \texttt{utility}(n) & \text{pro } n \text{ terminalni uzel} \\ \max_{s \in successors(n)} \texttt{eminimax} & \text{pro } n \text{ je MAX uzel} \\ \min_{s \in successors(n)} \texttt{eminimax} & \text{pro } n \text{ je MIN uzel} \\ \sum_{s \in successors(n)} P(s).\texttt{eminimax} & \text{pro } n \text{ je uzel nahody} \end{cases}$$

Hry s prvkem náhody

Aplikace klasické metody MINIMAXU, kdy MINIMAX hodnoty jsou nahrazeny očekávanými hodnotami – EMINMAX:



Hry s prvkem náhody

hod kostkou zvyšuje b – 21 možných hodů se 2 kostkami.
Backgammon má cca 20 legálních tahů.

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

se vzrůstající hloubkou klesá pravěpodobnost dosažení daného uzlu ⇒ význam predikce klesá

α/β prořezávání je velmi efektivní

$\mathrm{TDG\textsc{ammon}}$ prohledává do hloubky 2 a používá velmi dobrou funkci `eval` $\approx$ mistrovská úroveň