Hry a základní herní strategie

Aleš Horák

E-mail: hales@fi.muni.cz
http://nlp.fi.muni.cz/uui/

Obsah:

- Hry vs. Prohledávání stavového prostoru
- Algoritmus Minimax
- Algoritmus Alfa-Beta prořezávání
- Nedeterministické hry
- Hry s nepřesnými znalostmi

Hry × Prohledávání stavového prostoru

Multiagentní prostředí:

- agent musí brát v úvahu akce jiných agentů → jak ovlivní jeho vlastní prospěch
- vliv ostatních agentů – prvek náhody
- kooperativní × soupeřící multiagentní prostředí (MP)

Hry:

- matematická teorie her (odvětví ekonomie) – kooperativní i soupeřící MP, kde vliv všech agentů je *významný*
- hra v UI = obv. deterministické MP, 2 střídající se agenti, výsledek hry je vzájemně opačný nebo shoda

Algoritmy soupeřícího prohledávání (*adversarial search*):

- oponent dělá dopředu neurčitelné tahy → řešením je strategie, která počítá se všemi možnými tahy protivníka
- časový limit ⇒ zřejmě nenajdeme optimální řešení → hledáme lokálně optimální řešení

Hry a UI – historie

- Babbage, 1846 – počítač porovnává přínos různých herních tahů
- von Neumann, 1944 – algoritmy perfektní hry
- Zuse, Wiener, Shannon, 1945–50 – přibližné vyhodnocování
- Turing, 1951 – první šachový program (jen na papíře)
- Samuel, 1952–57 – strojové učení pro zpřesnění vyhodnocování
- McCarthy, 1956 – prořezávání pro možnost hlubšího prohledávání

Hry a UI – historie

- Babbage, 1846 – počítač porovnává přínos různých herních tahů
- von Neumann, 1944 – algoritmy perfektní hry
- Zuse, Wiener, Shannon, 1945–50 – přibližné vyhodnocování
- Turing, 1951 – první šachový program (jen na papíře)
- Samuel, 1952–57 – strojové učení pro zpřesnění vyhodnocování
- McCarthy, 1956 – prořezávání pro možnost hlubšího prohledávání

řešení her je zajímavým předmětem studia $\leftarrow$ je obtížné:
průměrný faktor větvení v šachách $b = 35$
pro 50 tahů 2 hráčů . . .

prohledávací strom $\approx 35^{100} \approx 10^{154}$ uzlů ($\approx 10^{40}$ stavů)

Hry a UI – aktuální výsledky

- dáma – 1994 program *Chinook* porazil světovou šampionku Marion Tinsley. Používá úplnou databázi tahů pro ≤ 8 figur (443 748 401 247 pozic).
- šachy – 1997 porazil stroj *Deep Blue* světového šampiona Gary Kasparova 3½–2½. Stroj počítá 200 mil. pozic/s, sofistikované vyhodnocování a nezveřejněné metody pro prozkoumávání některých tahů až do hloubky 40 tahů. 2006 porazil program *Deep Fritz* na PC světového šampiona Vladimíra Kramnika 2–4. V současnosti vyhrávají turnaje i programy na slabším hardware mobilních telefonů s 20 tis. pozic/s.
- Othello – světoví šampioni odmítají hrát s počítači, protože stroje jsou příliš dobré. Othello (též Reversi) pro dva hráče na desce 8×8 – snaží se mezi své dva kameny uzavřít soupeřovy, které se přebarví. Až se zaplní deska, spočítají se kameny.
- Go – do roku 2008 světoví šampioni odmítali hrát s počítači, protože stroje jsou příliš slabé. V Go je $b > 300$, takže počítače mohou používat téměř pouze znalostní bázi vzorových her.
  od 2009 – první programy dosahují pokročilejší amatérské úrovně (zejména na desce 9×9, nižší úroveň i na 19×19).

Typy her

	deterministické	*s náhodou*
perfektní znalosti	šachy, dáma, Go, Othello	backgammon, monopoly
nepřesné znalosti		bridge, poker, scrabble

Hledání optimálního tahu

2 hráči – MAX ($\triangle$) a MIN ($\bigtriangledown$)

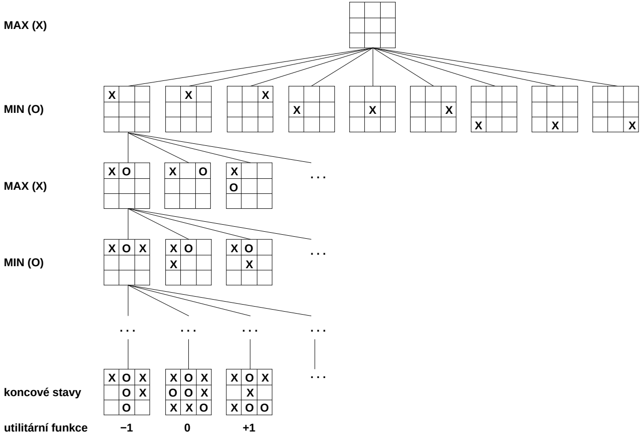
MAX je první na tahu a pak se střídají až do konce hry

hra = prohledávací problém:

- počáteční stav – počáteční herní situace + kdo je na tahu
- přechodová funkce – vrací dvojice (legální tah, výsledný stav)
- ukončovací podmínka – určuje, kdy hra končí, označuje koncové stavy
- utilitární funkce – numerické ohodnocení koncových stavů

Hledání optimálního tahu – pokrač.

počáteční stav a přechodová funkce definují herní strom:



Algoritmus Minimax

Hráč MAX ($\triangle$) musí *prohledat* herní strom pro zjištění nejlepšího tahu proti hráči MIN (∇)

$\rightarrow$ zjistit nejlepší hodnotu minimax – zajišťuje *nejlepší výsledek* proti *nejlepšímu protivníkovi*

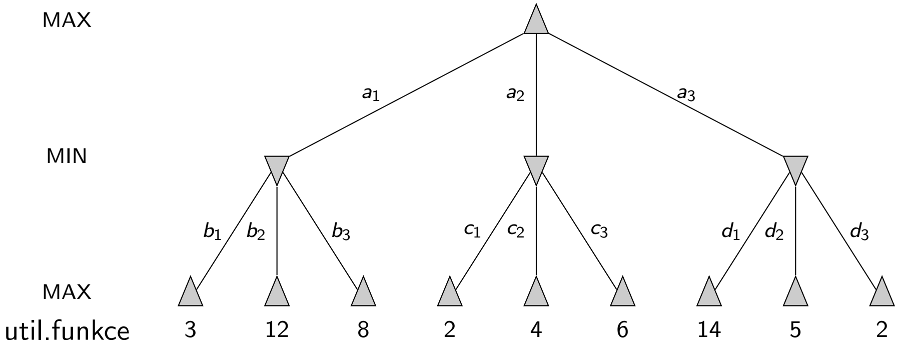
$$\text{Hodnota minimax}(n) = \begin{cases} \text{utility}(n), \\ \qquad \text{pro koncový stav } n \\ \max_{s \in \text{moves}(n)} \text{Hodnota minimax}(s), \\ \qquad \text{pro MAX uzel } n \\ \min_{s \in \text{moves}(n)} \text{Hodnota minimax}(s), \\ \qquad \text{pro MIN uzel } n \end{cases}$$

Algoritmus Minimax – pokrač.

příklad – hra jen na jedno kolo = 2 tahy (půlkola)

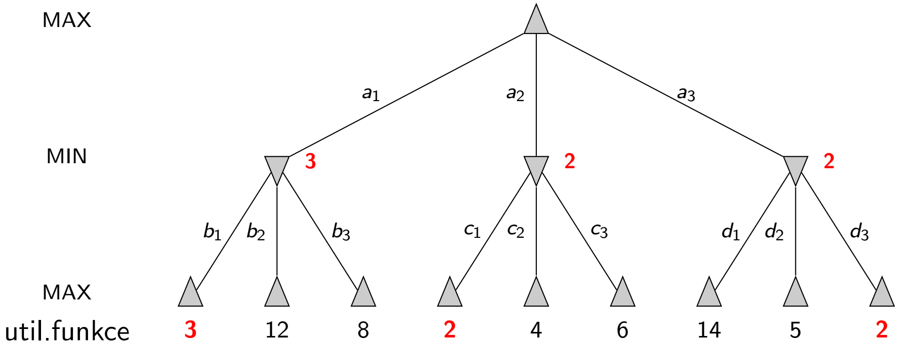
Algoritmus Minimax – pokrač.

příklad – hra jen na jedno kolo = 2 tahy (půlkola)



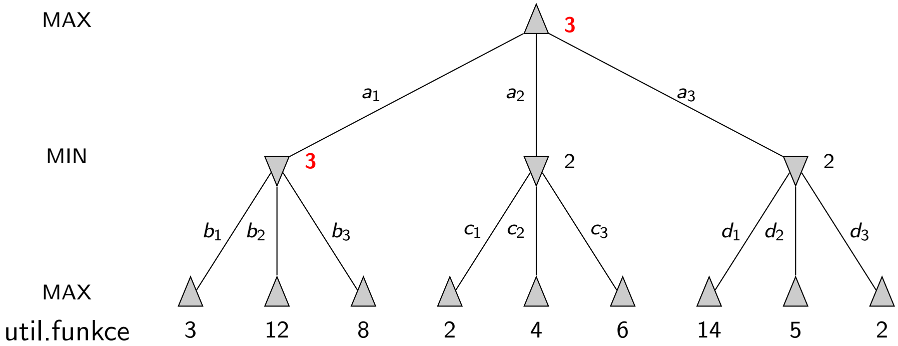
Algoritmus Minimax – pokrač.

příklad – hra jen na jedno kolo = 2 tahy (půlkola)



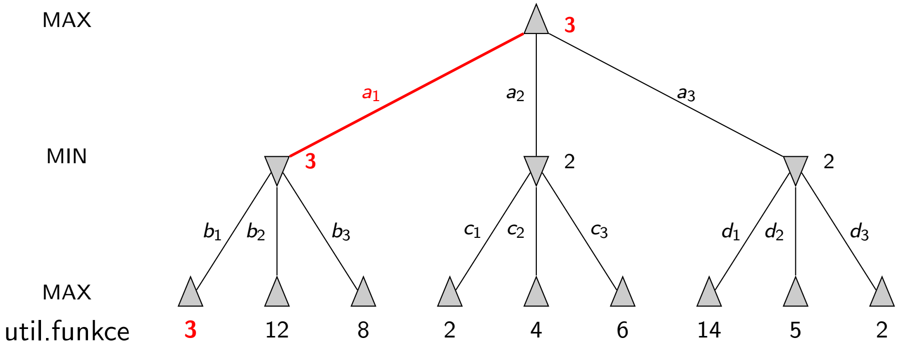
Algoritmus Minimax – pokrač.

příklad – hra jen na jedno kolo = 2 tahy (půlkola)



Algoritmus Minimax – pokrač.

příklad – hra jen na jedno kolo = 2 tahy (půlkola)



Algoritmus Minimax – pokrač.

```
% minimax( +Pos, −BestSucc, −Val):
% Pos  je  rozložení  figur,  Val je  minimaxová hodnota tohoto rozložení;
% nejlepší   tah  z  Pos vede do rozložení  BestSucc
minimax( Pos, BestSucc, Val)  :-
    moves( Pos, PosList), !,  %           PosList  je  seznam legálních  tahů z  Pos
    best( PosList,  BestSucc, Val)
    ;
    staticval(  Pos, Val).  %            Pos nemá následníky: ohodnotíme staticky

best( [Pos],  Pos, Val)  :- minimax( Pos, _,  Val), !.
best( [Pos1 | PosList],  BestPos, BestVal)  :-
    minimax( Pos1, _, Val1),
    best( PosList,  Pos2, Val2),
    betterof(  Pos1, Val1,  Pos2, Val2,  BestPos, BestVal).
betterof(  Pos0, Val0,  Pos1, Val1,  Pos0, Val0)  :- %   Pos0 je  lepší  než Pos1
    min_to_move( Pos0), %                        MIN na tahu v Pos0
    Val0 > Val1, !  %                            MAX chce nejvyšší hodnotu
    ;
    max_to_move( Pos0), %                        MAX na tahu v Pos0
    Val0 < Val1, !.  %                           MIN chce nejmenší hodnotu
betterof(  Pos0, Val0,  Pos1, Val1,  Pos1, Val1).  %    jinak  je  Pos1 lepší  než Pos0
```

Algoritmus Minimax – vlastnosti

úplnost
optimálnost
časová složitost
prostorová složitost

Algoritmus Minimax – vlastnosti

úplnost	úplný pouze pro konečné stromy
optimálnost	
časová složitost	
prostorová složitost	

Algoritmus Minimax – vlastnosti

úplnost	úplný pouze pro konečné stromy
optimálnost	je optimální proti optimálnímu oponentovi
časová složitost	
prostorová složitost	

Algoritmus Minimax – vlastnosti

úplnost	úplný pouze pro konečné stromy
optimálnost	je optimální proti optimálnímu oponentovi
časová složitost	$O(b^m)$
prostorová složitost	

Algoritmus Minimax – vlastnosti

úplnost	úplný pouze pro konečné stromy
optimálnost	je optimální proti optimálnímu oponentovi
časová složitost	$O(b^m)$
prostorová složitost	$O(bm)$, prohledávání do hloubky

Algoritmus Minimax – vlastnosti

úplnost	úplný pouze pro konečné stromy
optimálnost	je optimální proti optimálnímu oponentovi
časová složitost	$O(b^m)$
prostorová složitost	$O(bm)$, prohledávání do hloubky

šachy . . . $b \approx 35, m \approx 100 \Rightarrow$ přesné řešení není možné

např. $b^m = 10^6, b = 35 \Rightarrow m \approx 4$

4-tahy $\approx$ člověk-nováček
8-tahů $\approx$ člověk-mistr, typické PC
12-tahů $\approx$ Deep Blue, Kasparov

Časové omezení

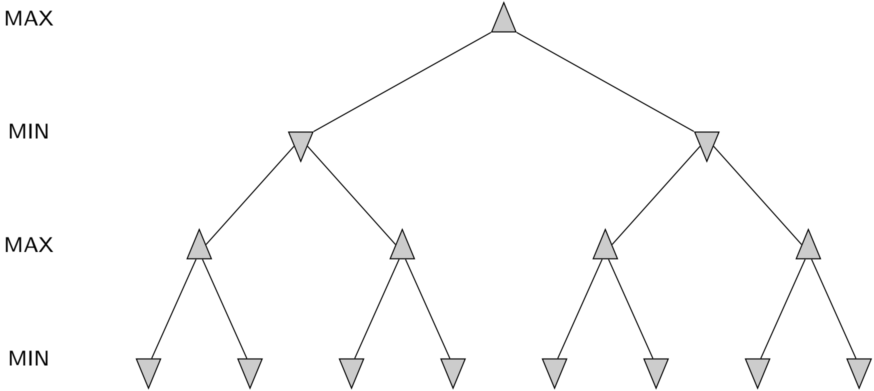
předpokládejme, že máme 100 sekund + prozkoumáme 10^4 uzlů/s
$\Rightarrow$ 10^6 uzlů na 1 tah

řešení **minimax_cutoff**:

- ohodnocovací funkce odhad přínosu pozice
  nahradí utilitární funkci
- ořezávací test (*cutoff test*) – např. hloubka nebo hodnota ohodnocovací funkce
  nahradí koncový test

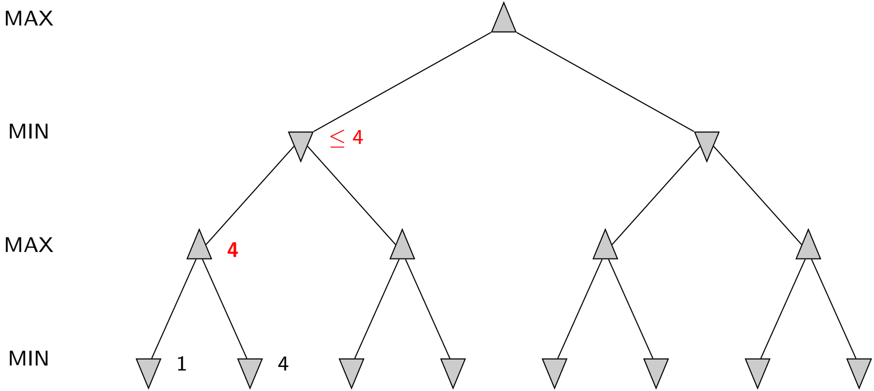
Algoritmus Alfa-Beta prořezávání

Příklad stromu, který zpracuje predikát **minmax**
Alfa-Beta odřízne expanzi některý uzlů ⇒ Alfa-Beta procedura je efektivnější variantou minimaxu



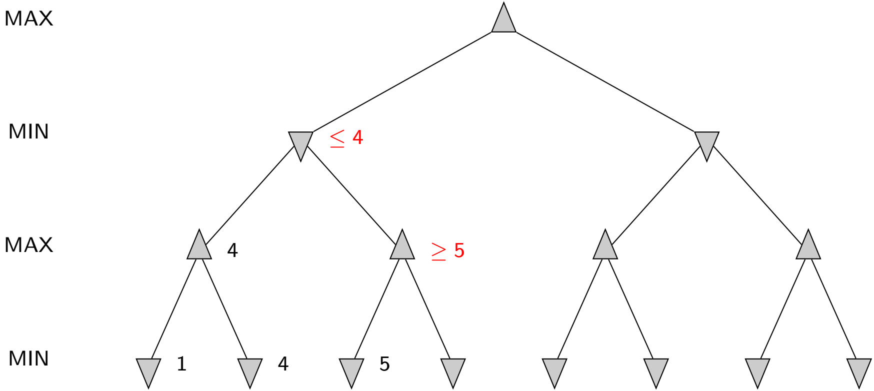
Algoritmus Alfa-Beta prořezávání

Příklad stromu, který zpracuje predikát **minmax**
Alfa-Beta odřízne expanzi některý uzlů $\Rightarrow$ Alfa-Beta procedura je efektivnější variantou minimaxu



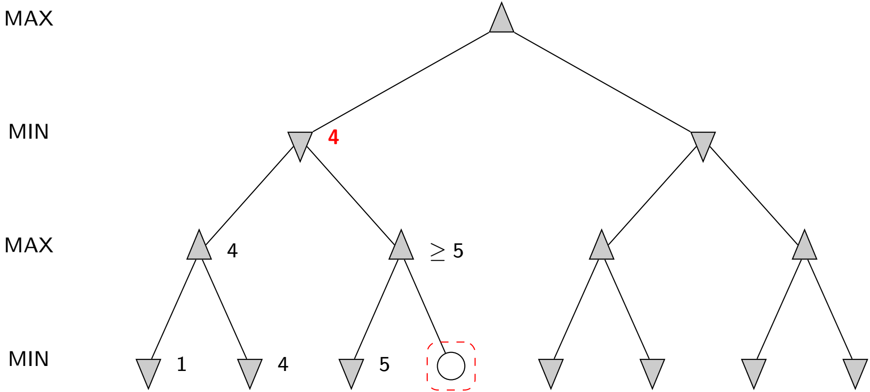
Algoritmus Alfa-Beta prořezávání

Příklad stromu, který zpracuje predikát **minmax**
Alfa-Beta odřízne expanzi některý uzlů ⇒ Alfa-Beta procedura je efektivnější variantou minimaxu



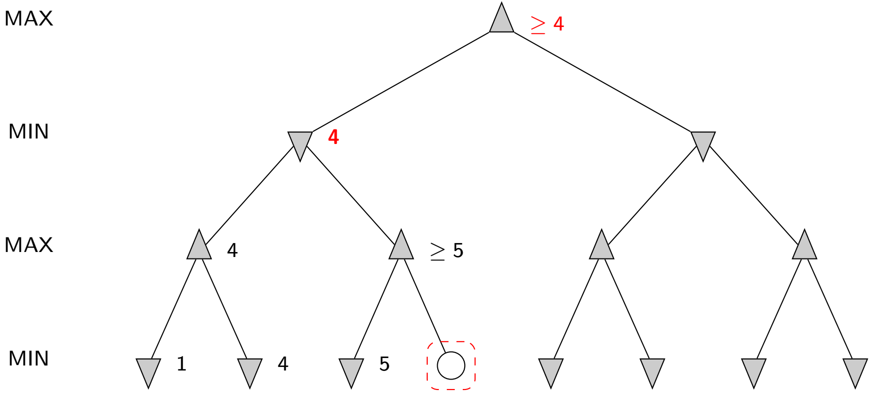
Algoritmus Alfa-Beta prořezávání

Příklad stromu, který zpracuje predikát **minmax**
Alfa-Beta odřízne expanzi některý uzlů ⇒ Alfa-Beta procedura je efektivnější variantou minimaxu



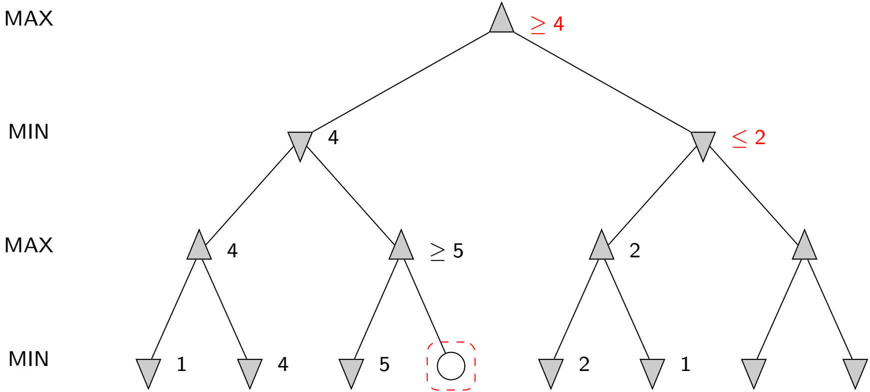
Algoritmus Alfa-Beta prořezávání

Příklad stromu, který zpracuje predikát **minmax**
Alfa-Beta odřízne expanzi některý uzlů ⇒ Alfa-Beta procedura je efektivnější variantou minimaxu



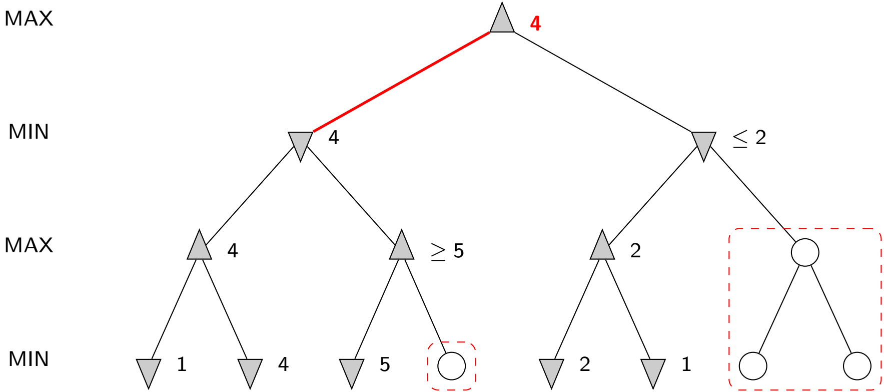
Algoritmus Alfa-Beta prořezávání

Příklad stromu, který zpracuje predikát **minmax**
Alfa-Beta odřízne expanzi některý uzlů ⇒ Alfa-Beta procedura je efektivnější variantou minimaxu



Algoritmus Alfa-Beta prořezávání

Příklad stromu, který zpracuje predikát **minmax**
Alfa-Beta odřízne expanzi některý uzlů ⇒ Alfa-Beta procedura je efektivnější variantou minimaxu



Algoritmus Alfa-Beta prořezávání – vlastnosti

- prořezávání neovlivní výsledek $\Rightarrow$ je stejný jako u minimaxu
- dobré uspořádání přechodů (možných tahů) ovlivní efektivitu prořezávání
- v případě "nejlepšího" uspořádání časová složitost$= O(b^{m/2})$
  $\Rightarrow$ zdvojí hloubku prohledávání
  $\Rightarrow$ může snadno dosáhnout hloubky 8 v šachu, což už je použitelná úroveň

Algoritmus Alfa-Beta prořezávání – vlastnosti

- prořezávání neovlivní výsledek ⇒ je stejný jako u minimaxu
- dobré uspořádání přechodů (možných tahů) ovlivní efektivitu prořezávání
- v případě "nejlepšího" uspořádání časová složitost$= O(b^{m/2})$
  ⇒ zdvojí hloubku prohledávání
  ⇒ může snadno dosáhnout hloubky 8 v šachu, což už je použitelná úroveň

označení $\alpha - \beta$:

- $\alpha \ldots$ doposud nejlepší hodnota pro MAXe
- $\beta \ldots$ doposud nejlepší hodnota pro MINa
- $<\alpha, \beta> \ldots$ interval ohodnocovací funkce v průběhu výpočtu (na začátku $<-\infty, \infty>$)
-

minimax ... $V(P)$	$\alpha - \beta \ldots V(P, \alpha, \beta)$
když $V(P) \leq \alpha$	$V(P, \alpha, \beta) = \alpha$
když $\alpha < V(P) < \beta$	$V(P, \alpha, \beta) = V(P)$
když $V(P) \geq \beta$	$V(P, \alpha, \beta) = \beta$

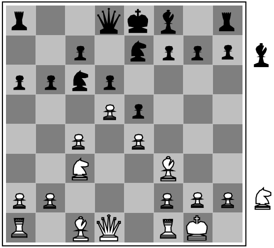
Algoritmus Alfa-Beta prořezávání

```
alphabeta( Pos, Alpha, Beta, GoodPos, Val)  :- moves( Pos, PosList),  !,
    boundedbest( PosList, Alpha, Beta, GoodPos, Val)
    ; staticval(   Pos, Val).  %                          statické   ohodnocení Pos

boundedbest( [Pos | PosList],  Alpha, Beta, GoodPos, GoodVal) :-
    alphabeta( Pos, Alpha, Beta, _,  Val),
    goodenough( PosList, Alpha, Beta, Pos, Val,  GoodPos, GoodVal).

goodenough( [], _,  _,  Pos, Val, Pos, Val)   :- !.    %             nejsou  další   kandidáti
goodenough( _, Alpha, Beta, Pos, Val,  Pos, Val)   :-
    min_to_move( Pos), Val > Beta, !  %             MAX dosáhl horní hranici
    ; max_to_move( Pos), Val < Alpha, !.  %                      MIN dosáhl dolní  hranici
goodenough( PosList, Alpha, Beta, Pos, Val,  GoodPos, GoodVal) :-
    newbounds( Alpha, Beta, Pos, Val, NewAlpha, NewBeta), % uprav hranice
    boundedbest( PosList, NewAlpha, NewBeta, Pos1, Val1),
    betterof(  Pos, Val,  Pos1, Val1,  GoodPos, GoodVal).

newbounds( Alpha, Beta, Pos, Val,  Val,  Beta)   :-
    min_to_move( Pos), Val > Alpha, !.  %                        MAX zvýšil dolní  hranici
newbounds( Alpha, Beta, Pos, Val, Alpha,  Val)   :-
    max_to_move( Pos), Val < Beta, !. %                          MIN snížil  horní  hranici
newbounds( Alpha, Beta, _, _,  Alpha,  Beta).  %                 jinak  hranice  nezměněny

betterof(  Pos, Val,  Pos1, Val1,  Pos, Val)   :- min_to_move( Pos),  Val > Val1, !
    ; max_to_move( Pos), Val < Val1, !.  %              Pos je  lepší   než Pos1
betterof(  _,  _,  Pos1, Val1,  Pos1, Val1).  %                  jinak  je  lepší   Pos1
```

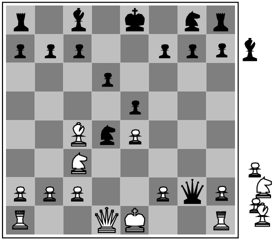
Možnosti vylepšení Minimax/Alpha-Beta

- vyhodnocovat pouze klidné stavy (quiescent search)
- při vyhodnocování počítat s efektem horizontu – zvraty mimo prohledanou oblast
- dopředné ořezávání – některé stavy se ihned zahazují
  bezpečné např. pro symetrické tahy nebo pro tahy hluboko ve stromu

Ohodnocovací funkce

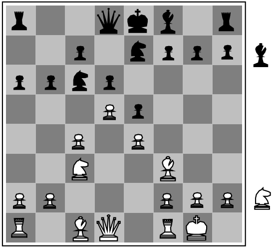


Černý na tahu
Bílý ma o něco lepší pozici

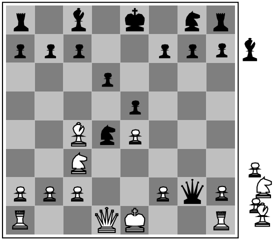


Bílý na tahu
Černý vítězí

Ohodnocovací funkce



Černý na tahu
Bílý ma o něco lepší pozici



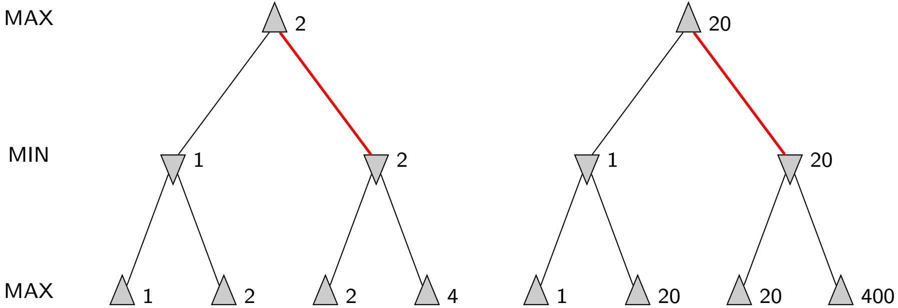
Bílý na tahu
Černý vítězí

Pro šachy typicky lineární vážený součet rysů

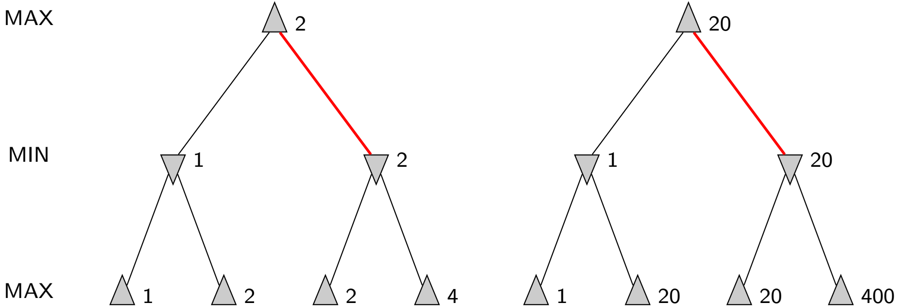
$$Eval(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s) = \sum_{i=1}^{n} w_i f_i(s)$$

např. $w_1 = 9$
$f_1(s) =$ (počet bílých královen) – (počet černých královen)
...

Ohodnocovací funkce – odchylky



Ohodnocovací funkce – odchylky



chová se stejně pro libovolnou monotónní transformaci funkce *Eval*
záleží pouze na uspořádání → ohodnocení v deterministické hře funguje jako ordinální funkce

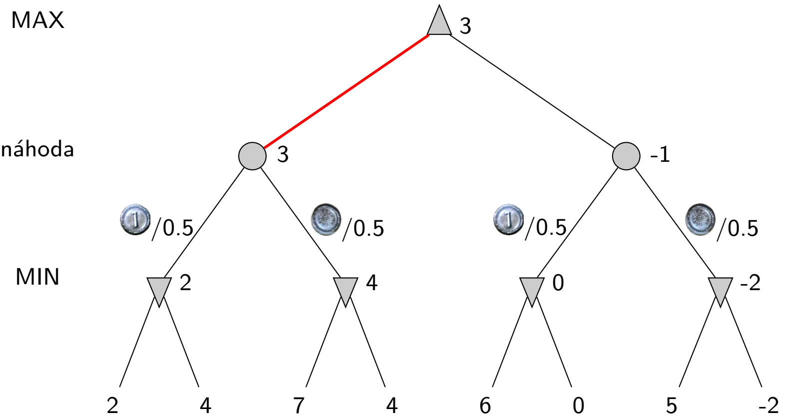
Nedeterministické hry

náhoda $\longleftarrow$ hod kostkou, hod mincí, míchání karet

Nedeterministické hry

náhoda ⟵ hod kostkou, hod mincí, míchání karet

příklad – 1 tah s házením mincí:



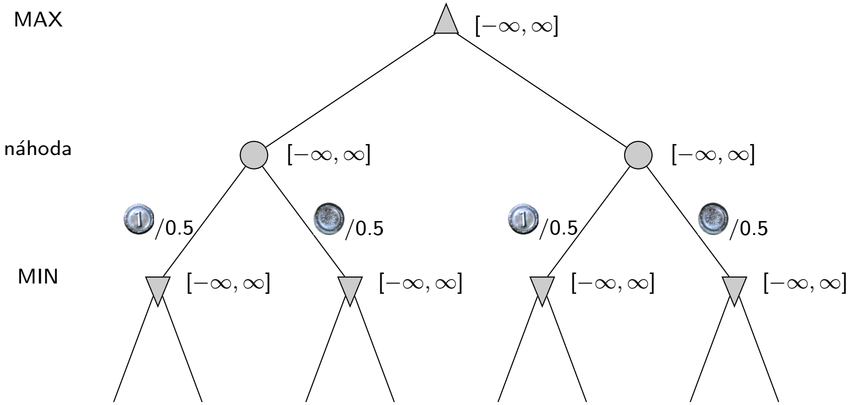
Algoritmus Minimax pro nedeterministické hry

expect_minimax . . . počítá perfektní hru s přihlédnutím k náhodě
rozdíl je pouze v započítání uzlů *náhoda*:

$$\text{expect\_minimax}(n) = \begin{cases} \text{utility}(n) \\ \qquad \text{pro koncový stav } n \\ \max_{s \in \text{moves}(n)} \text{expect\_minimax}(s) \\ \qquad \text{pro MAX uzel } n \\ \min_{s \in \text{moves}(n)} \text{expect\_minimax}(s) \\ \qquad \text{pro MIN uzel } n \\ \sum_{s \in \text{moves}(n)} P(s) \cdot \text{expect\_minimax}(s) \\ \qquad \text{pro uzel náhody } n \end{cases}$$

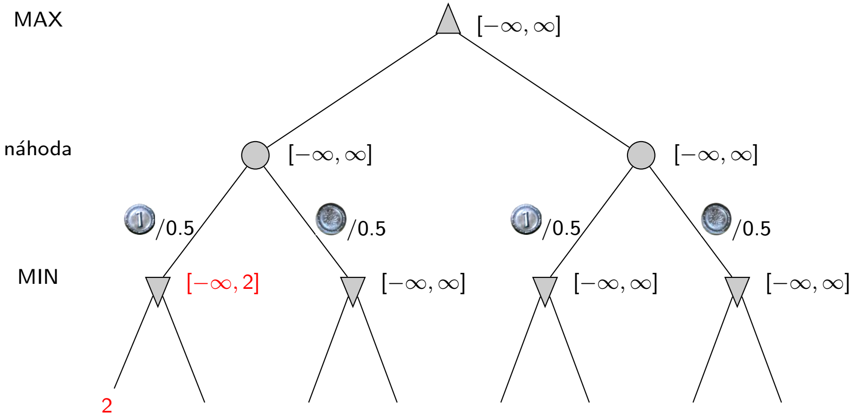
Prořezávání v nedeterministických hrách

je možné použít upravené Alfa-Beta prořezávání



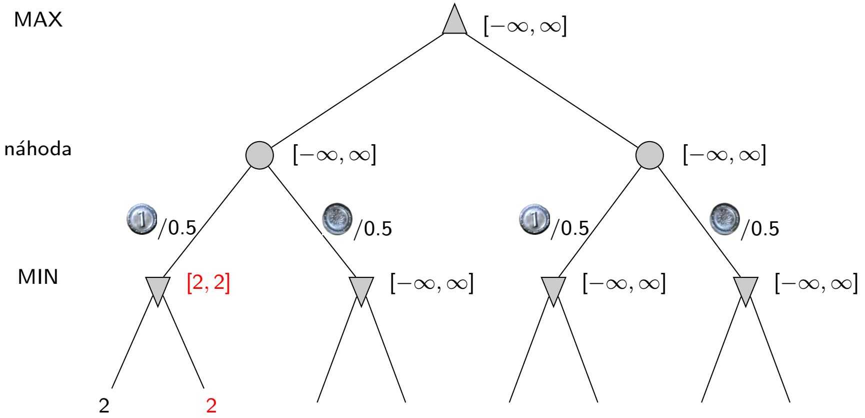
Prořezávání v nedeterministických hrách

je možné použít upravené Alfa-Beta prořezávání



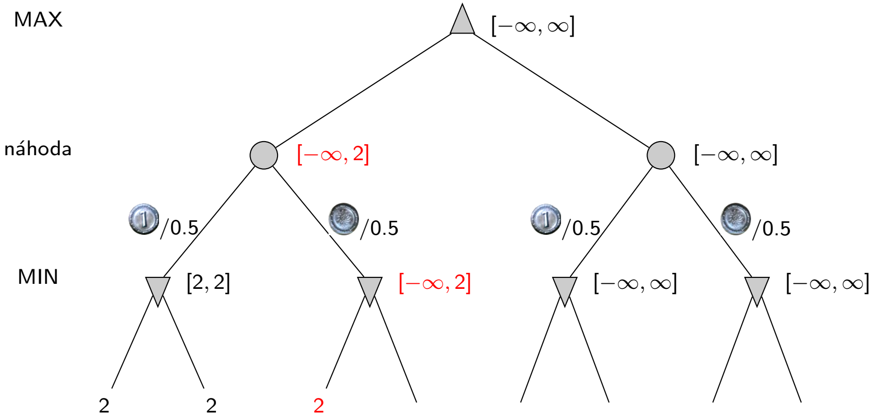
Prořezávání v nedeterministických hrách

je možné použít upravené Alfa-Beta prořezávání



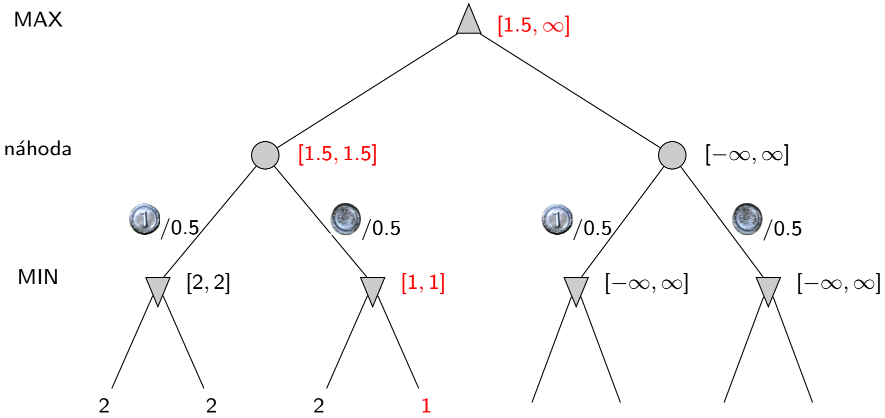
Prořezávání v nedeterministických hrách

je možné použít upravené Alfa-Beta prořezávání



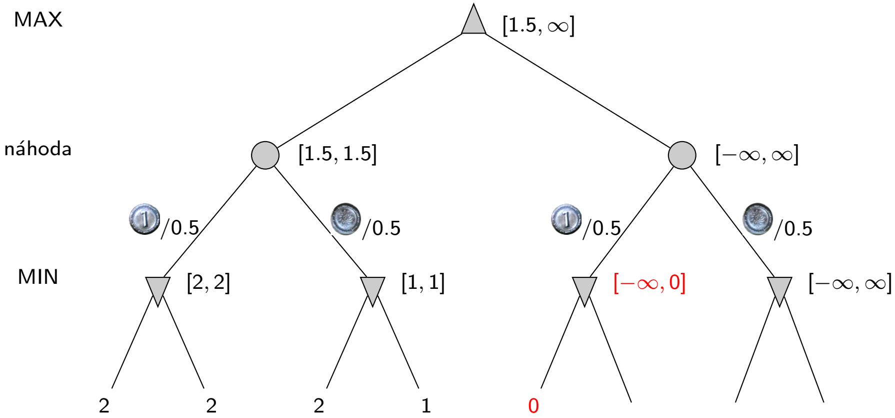
Prořezávání v nedeterministických hrách

je možné použít upravené Alfa-Beta prořezávání



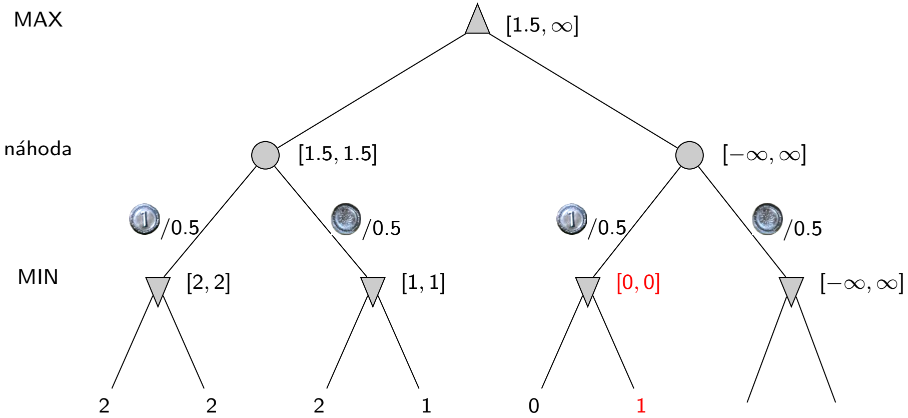
Prořezávání v nedeterministických hrách

je možné použít upravené Alfa-Beta prořezávání



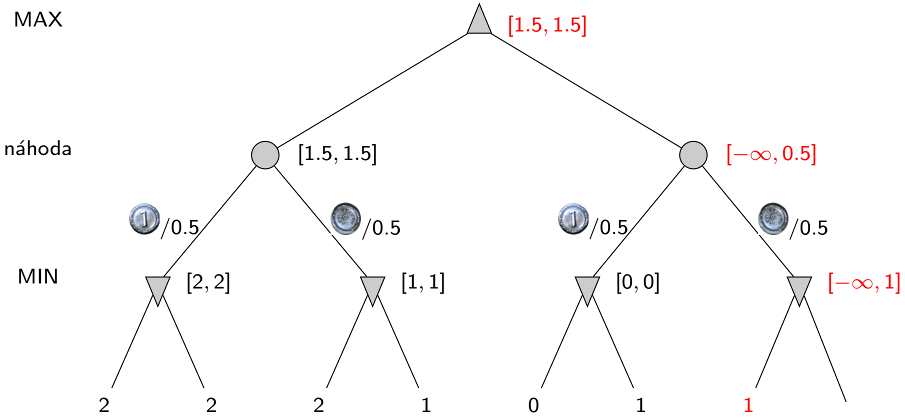
Prořezávání v nedeterministických hrách

je možné použít upravené Alfa-Beta prořezávání



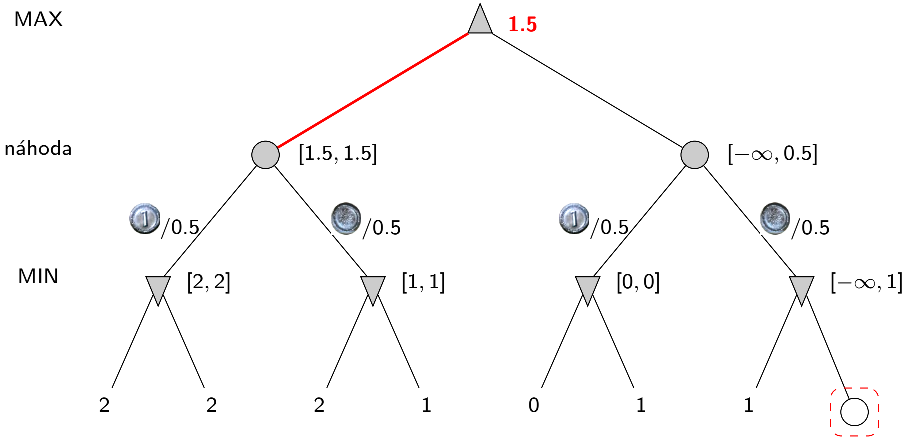
Prořezávání v nedeterministických hrách

je možné použít upravené Alfa-Beta prořezávání



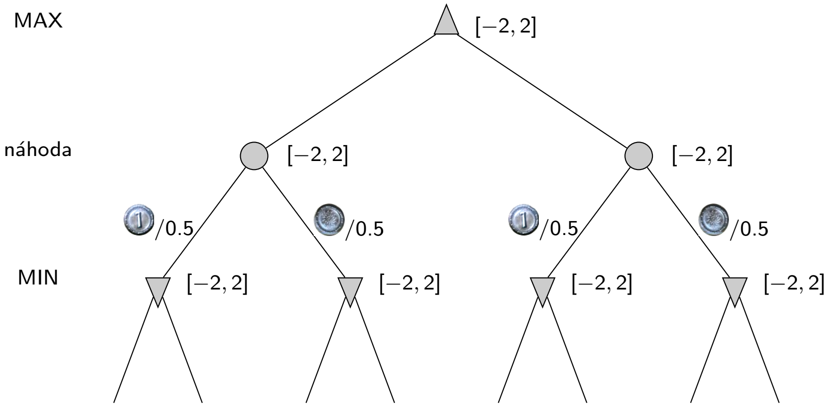
Prořezávání v nedeterministických hrách

je možné použít upravené Alfa-Beta prořezávání



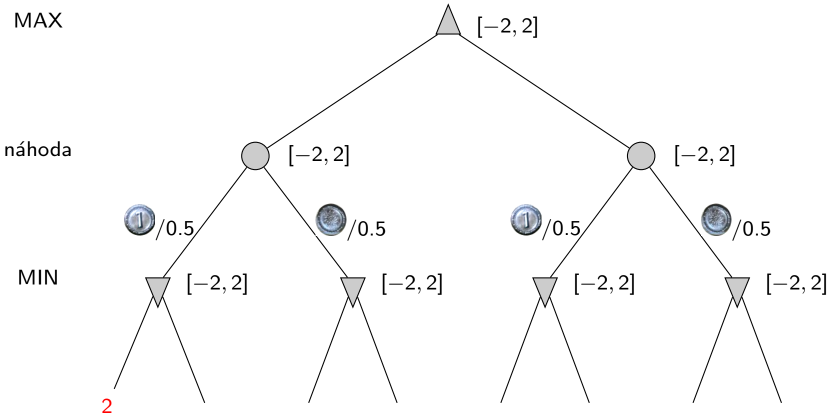
Prořezávání v nedeterministických hrách – pokrač.

pokud je možno dopředu stanovit limity na ohodnocení listů → ořezávání je větší



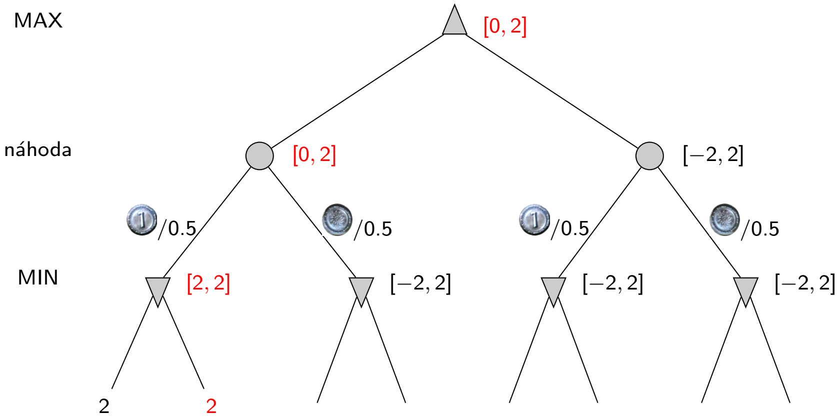
Prořezávání v nedeterministických hrách – pokrač.

pokud je možno dopředu stanovit limity na ohodnocení listů $\rightarrow$ ořezávání je větší



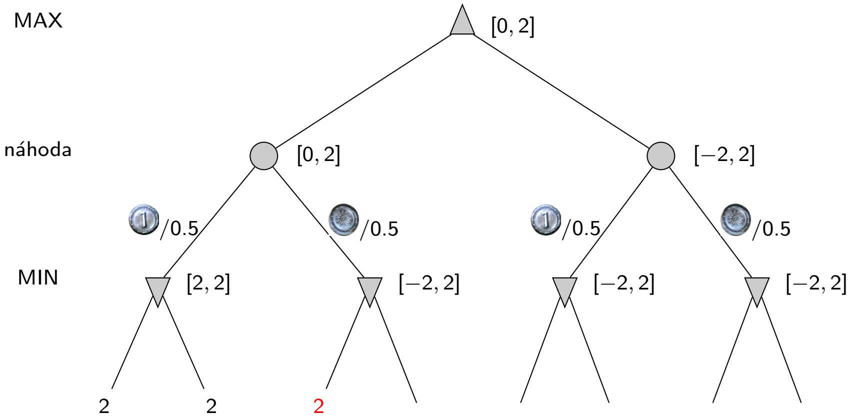
Prořezávání v nedeterministických hrách – pokrač.

pokud je možno dopředu stanovit limity na ohodnocení listů $\rightarrow$
ořezávání je větší



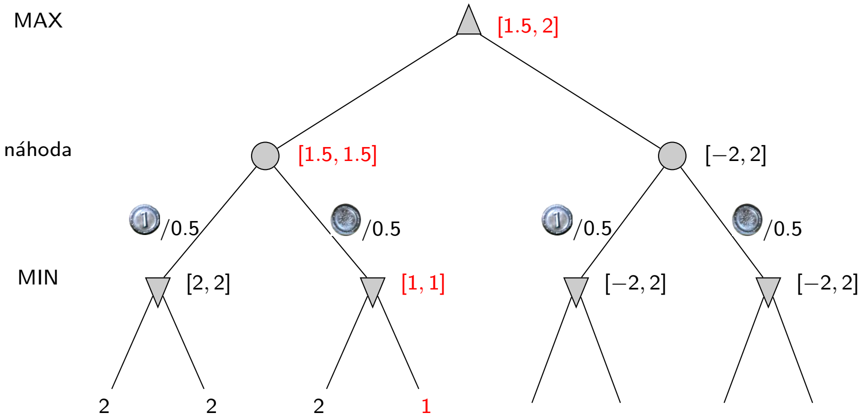
Prořezávání v nedeterministických hrách – pokrač.

pokud je možno dopředu stanovit limity na ohodnocení listů $\rightarrow$ ořezávání je větší



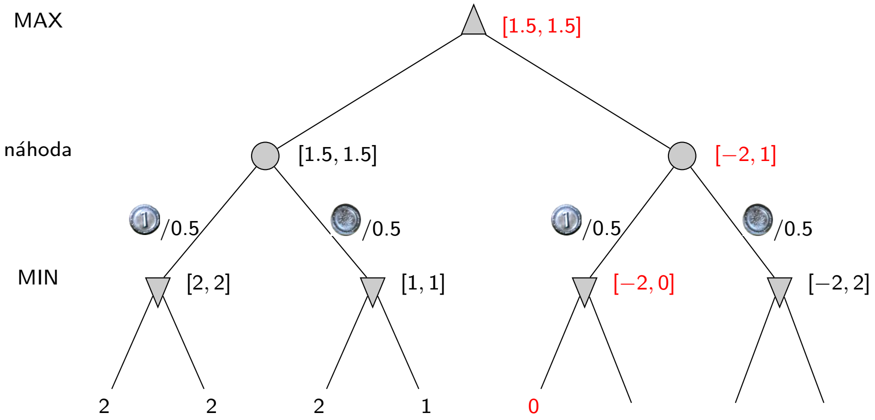
Prořezávání v nedeterministických hrách – pokrač.

pokud je možno dopředu stanovit limity na ohodnocení listů → ořezávání je větší



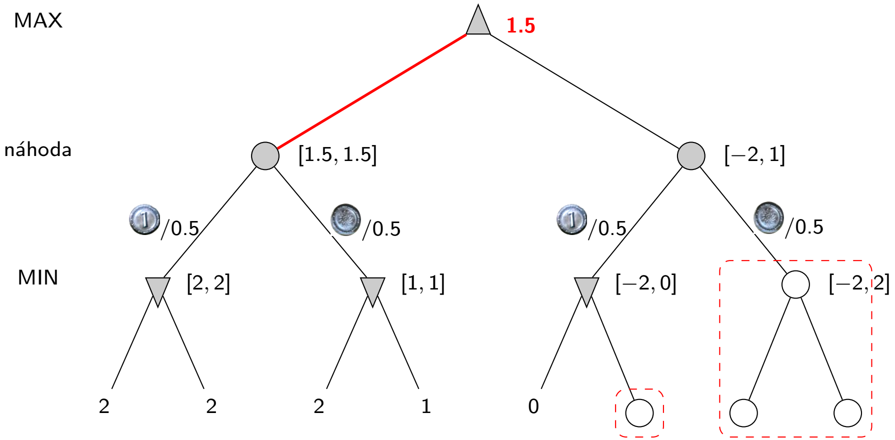
Prořezávání v nedeterministických hrách – pokrač.

pokud je možno dopředu stanovit limity na ohodnocení listů $\rightarrow$ ořezávání je větší



Prořezávání v nedeterministických hrách – pokrač.

pokud je možno dopředu stanovit limity na ohodnocení listů → ořezávání je větší



Nedeterministické hry v praxi

- hody kostkou zvyšují b $\rightarrow$ se dvěma kostkami 21 možných výsledků
- backgammon – 20 legálních tahů:

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

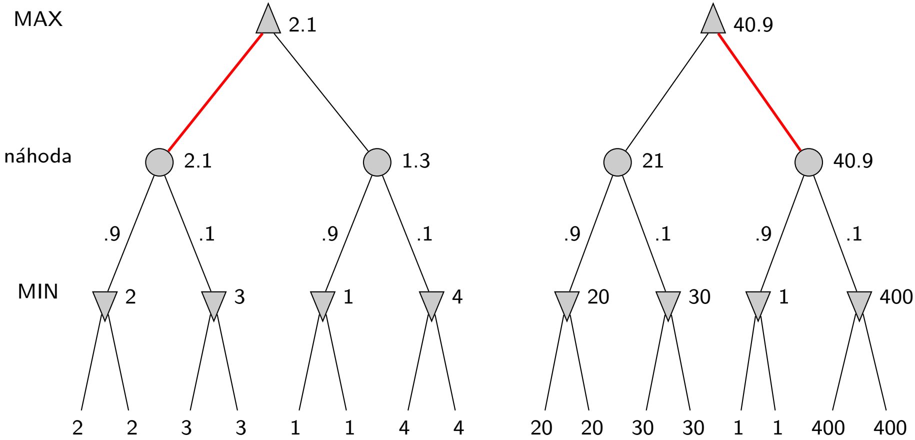
- jak se zvyšuje hloubka $\rightarrow$

  pravděpodobnost dosažení zvoleného uzlu klesá

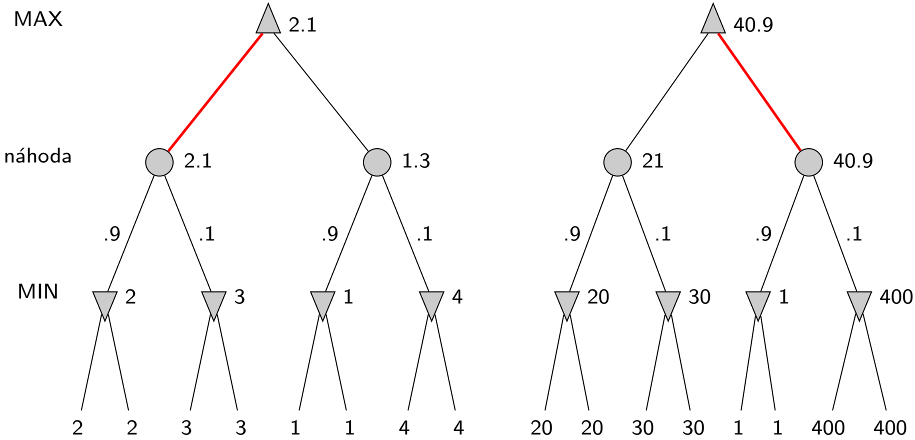
  $\Rightarrow$ význam prohledávání se snižuje
- alfa-beta prořezávání je mnohem méně efektivní
- program TDGammon používá prohledávání do hloubky 2 + velice dobrou *Eval* funkci

  $\approx$ dosahuje úrovně světového šampionátu

Odchylka v ohodnocení nedeterministických her



Odchylka v ohodnocení nedeterministických her



chování je zachováno pouze pro pozitivní lineární transformaci funkce *Eval*

Eval u nedeterministických her by tedy měla proporcionálně odpovídat očekávanému výnosu

Hry s nepřesnými znalostmi

- např. karetní hry $\rightarrow$ neznáme počáteční namíchání karet oponenta
- obvykle můžeme spočítat pravděpodobnost každého možného rozdání
- zjednodušeně – jako jeden velký hod kostkou na začátku
- prohledáváme ovšem ne reálný stavový prostor, ale domnělý stavový prostor
- program *Jack*, nejčastější vítěz počítačových šampionátů v bridgi:
  1. generuje 100 rozdání karet konzistentních s daným podáním
  2. vybírá akci, která je v průměru nejlepší

  V roce 2006 porazil Jack na soutěži 3 ze 7 top holandských hráčských párů.