

# Úvod do umělé inteligence (NAIL120)

9. cvičení

Jirka Fink

<https://ktiml.mff.cuni.cz/~fink/>

Katedra teoretické informatiky a matematické logiky  
Matematicko-fyzikální fakulta  
Univerzita Karlova v Praze

Letní semestr 2023/24

Poslední změna 23. dubna 2024

Licence: Creative Commons BY-NC-SA 4.0

### Zadání (zkráceno)

- Na Marsu přistane robot, který se má dostat na základnu
- Přistání není 100 % úspěšné, takže
  - nepřistál přímo na základně, ale musí k ní dojet
  - poškodili se motory a robot často jede jinam než řídicí jednotka zadala
- Naštěstí funguje alespoň lokalizace, takže robot vždy ví, kde se na čtvercové mřížce nachází
- Při přesunu používá drahocennou energii
- Pro každou pozici je známo, kolik energie je zapotřebí k projetí
- Cílem je najít nejkratší cestu na základnu
  - vzhledem k poškozeným motorům nelze předem spočítat optimální cestu
  - proto pro každou pozici spočítáme nejlepší pokyn, který může řídicí jednotka zadat

### Postup

- Nejprve si představte, že pozice a přechodové akce tvoří acyklický graf, a vytvořte odpovídající Bellmanovu funkci užitku
- Implementujte jednodušší algoritmus založený jen na „value update“ a zkontrolujte si, že na pomocných testech dává správné řešení
- K implementaci „policy update“ si napište příslušnou soustavu rovnic

## Popis

Teorie her analyzuje široké spektrum konfliktních rozhodovacích situací, kde může docházet ke střetu zájmů.

## Typy her

- Kooperativní/kompetitivní: Můžou hráči spolupracovat a vytvářet koalice?
- Simultánní/tahové: Hrají hráči současně?
- S úplnou informací: Mají hráči všechny informace?
- Stochastické/deterministické: Ovlivňuje výsledek hry náhodný jev?
- Konečné: Je hra vždy konečná?

## Příklady her

- Šachy, go, poker
- Aukce
- Maticové hry

## Triviální hra (NIM)

- Máme hromádku  $n$  sirek, které dva hráči postupně odebírají
- Hráč může odebrat 1, 2 nebo 3 sirky
- Vyhrává hráč, který odebere poslední sirku
- Jakou strategii má hráč zvolit?

## Pojmy

- Stav: Úplný popis situace hry
- Počáteční a koncový stav
- Pro koncové stavy pravidla určují, kdo vyhrává
- Stav je vyhrávající pro daného hráče, pokud může hrát tak, aby nakonec vyhrál

## Hry s užitkem

- Pro každý konečný stav je dáno, jaký užitek má který hráč
- Každý hráč se snaží hrát tak, aby maximalizoval svůj užitek
- Hry s nulovým součtem: Součet všech užiteků v každém koncovém stavu je nulový

## NIM 2

- Máme hromádku  $n$  sirek, které dva hráči postupně odebírají
- Počet odebraných sirek musí být Fibonacciho číslo (například)
- Vyhrává hráč, který odebere poslední sirku
- Jak rozhodnout, zda je daný stav vyhrávající?

## Zpětné prořezávání

```
1 winning[0 ... n] = False
2 for i from 1 to n do
3   for Fibonacci numbers  $j \leq i$  do
4     if not winning[i-j] then
5       winning[i] = True
6       break # Terminate the inner loop
```

## Výhody a nevýhody při obecném použití

- Každý stav je vyhodnocen nejvýše jednou
- V netriviálních hrách nám nebude stačit paměť
- Nemusí být jednoduché určit všechny vyhrávající koncové stavy
- Nemusíme umět procházet stavy ve zpětném pořadí
  - Např. nalezení předchozího stavu v *Conway's Game of Life* je NP-těžké

## Dopředné prořezávání

```
1 Procedure is_winning(n)  
2   for Fibonacci numbers  $i \leq n$  do  
3     if not is_winning(n-i) then  
4       return True  
5   return False
```

## Výhody a nevýhody při obecném použití

- Vyžaduje málo paměti
- Některé stavy mohou být opakovaně prohledávány
- V netriviálních hrách se výsledku nedožijeme

## Příklad

První hráč odebírá Fibonacciho čísla, druhý mocniny dvojky

```
1 Procedure is_first_player_winning(n)
2   for Fibonacci numbers  $i \leq n$  do
3     if not is_second_player_winning( $n-i$ ) then
4       return True
5   return False
6 Procedure is_second_player_winning(n)
7   for i powers of 2 st.  $i \leq n$  do
8     if not is_first_player_winning( $n-i$ ) then
9       return True
10  return False
```



## Jak souvisí předchozí algoritmus s přednáškou?

- Na přednášce byl obecných min-max algoritmus
- Na předchozím slajdu žádný min-max není
- Funkce pro oba hráče jsou stejné (až na vyhodnocování odlišných pravidel)
- Máme min-max algoritmus nebo něco úplně jiného?
- Kde používáme alpha-beta prořezávání?

## Obecné převedení minimalizace na maximalizaci

Pro každou funkci  $f : M \rightarrow \mathbb{R}$ , kde  $M$  je libovolná množina platí

$$\min_{x \in M} f(x) = - \max_{x \in M} -f(x)$$

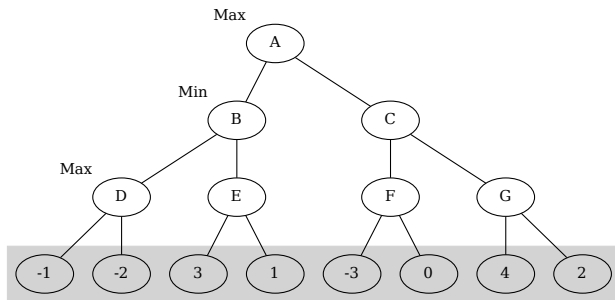
$$\max_{x \in M} f(x) = - \min_{x \in M} -f(x).$$

Tedy  $\max(\min(\max(\min(\max \dots u(x)))) = \max(- \max(- \max(- \max(- \max \dots u(x))))$

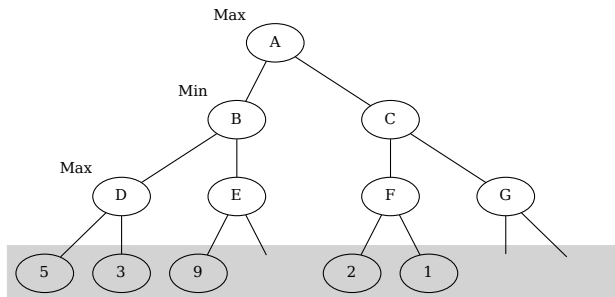
- $u(x)$  značí užitek koncových stavů
- Liché členy maximalizují užitek prvního hráče
- Sudé členy maximalizují užitek druhého hráče

## Jednoduchá půlící hrana

- Máme pole čísel velikosti  $2^n$
- Hráči se střídají
- Hráč vybere horní nebo dolní polovinu pole a druhou zahodí
- Po  $n$  tazích zůstane jedno číslo udávající výhru prvního hráče i ztrátu druhého
- Popište postup popisující vyhrávající strategii obou hráčů



Potřebujete k nalezení vyhrávající strategie znát ohodnocení všech koncových stavů?



**function** ALPHA-BETA-SEARCH(*game*, *state*) **returns** an action

player  $\leftarrow$  *game*.TO-MOVE(*state*)

value, move  $\leftarrow$  MAX-VALUE(*game*, *state*,  $-\infty$ ,  $+\infty$ )

**return** move

**function** MAX-VALUE(*game*, *state*,  $\alpha$ ,  $\beta$ ) **returns** a (utility, move) pair

**if** *game*.IS-TERMINAL(*state*) **then return** *game*.UTILITY(*state*, player), null

*v*  $\leftarrow -\infty$

**for each** *a* **in** *game*.ACTIONS(*state*) **do**

*v2*, *a2*  $\leftarrow$  MIN-VALUE(*game*, *game*.RESULT(*state*, *a*),  $\alpha$ ,  $\beta$ )

**if** *v2* > *v* **then**

*v*, *move*  $\leftarrow$  *v2*, *a*

$\alpha \leftarrow$  MAX( $\alpha$ , *v*)

**if** *v*  $\geq$   $\beta$  **then return** *v*, *move*

**return** *v*, *move*

**function** MIN-VALUE(*game*, *state*,  $\alpha$ ,  $\beta$ ) **returns** a (utility, move) pair

**if** *game*.IS-TERMINAL(*state*) **then return** *game*.UTILITY(*state*, player), null

*v*  $\leftarrow +\infty$

**for each** *a* **in** *game*.ACTIONS(*state*) **do**

*v2*, *a2*  $\leftarrow$  MAX-VALUE(*game*, *game*.RESULT(*state*, *a*),  $\alpha$ ,  $\beta$ )

**if** *v2* < *v* **then**

*v*, *move*  $\leftarrow$  *v2*, *a*

$\beta \leftarrow$  MIN( $\beta$ , *v*)

**if** *v*  $\leq$   $\alpha$  **then return** *v*, *move*

**return** *v*, *move*

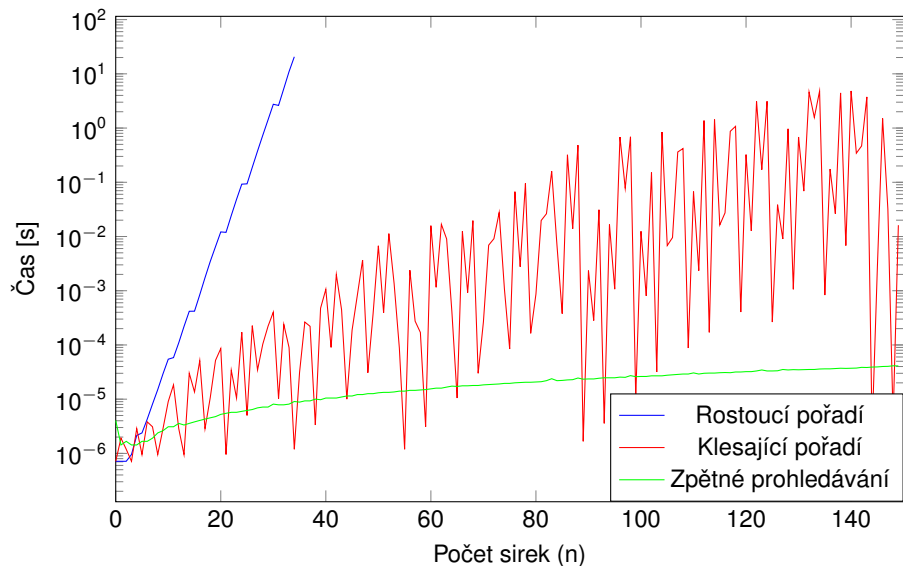
## Nejprve prozkoumat nadějnější stavy

- Použít heuristiku ohodnocující stavy
- Nejprve prozkoumáme stavy, které vedou na méně možných tahů druhého hráče

## Jak jednoduše urychlit tento algoritmus?

Oba hráči odebírají Fibonacciho čísla

```
1 Procedure is_winning(n)  
2   for Fibonacci numbers  $i \leq n$  do  
3     if not is_winning( $n-i$ ) then  
4       return True  
5   return False
```



## Předpočítat si ohodnocení některých stavů

- Předpočítat si stavy blízko koncových pomocí zpětné prohledávání
- Analýzou hry dokázat ohodnocení některých stavů

## Příklad

- Ve hře jsou 3 hromádky sirek
- Hráč může z jedné hromádky odebrat libovolný kladný počet sirek
- Vyhrává hráč, který odebral poslední sirku

Pro které stavy dokážete rovnou říct výsledek?

## Kešování prozkoumaných stavů

- Opakovaným výpočtům zabráníme kešováním výsledků vyhodnocených stavů
- Kešování je přínosně jen když často dochází k opakovanému vyhodnocování stavů
- Pamatováním výsledků všech stavů rychle dojde paměť

## Rychlejší nebo efektivnější heuristika

- Efektivnější heuristika může snížit počet prozkoumaných stavů
- Rychlejší heuristika může prozkoumat více stavů za stejný čas
- Heuristiky je nutné experimentálně porovnat, abychom zjistili, která je lepší



## Terminologie

- **Dominantní strategie hráče:** Nejlepší strategie pro hráče, ať ostatní udělají cokoliv
- **Nash equilibrium:** Kombinace strategií hráčů, kde se žádnému hráči samostatně nevyplatí svou strategii změnit
- **Pareto optimální:** Kombinace strategií hráčů taková, že neexistuje jiná kombinace strategií hráčů, ve které si nikdo nepohorší a alespoň jeden si polepší

## Vězňovo dilema

	2. zloděj neudá komplice	2. zloděj udá komplice
1. zloděj neudá komplice	(1,1)	(10,0)
1. zloděj udá komplice	(0,10)	(8,8)

Značení  $(a, b)$ :  $a$  je počet let ve vězení pro prvního zloděje a  $b$  pro druhého zloděje  
Existují dominantní, Nash equilibrium a Pareto optimální strategie?

Kluk a holka by chtěli strávit společné odpoledne. Holka by raději šla do kina, kluk na fotbal, ale oba by raději byli spolu.

	holka jde do kina	holka jde na fotbal
kluk jde do kina	(5,6)	(1,1)
kluk jde na fotbal	(2,2)	(6,5)

První číslo udává užitek pro kluka, druhé pro holku.

Existují dominantní, Nash equilibrium a Pareto optimální strategie?

- Proti sobě jedou dvě auta.
- Oba řidiči se můžou rozhodnout, zda pojedou v pravém či levém jízdním pruhu.
- Srážka oběma řidičům způsobí mnohem větší škody než když se vyhnou.
- Existují dominantní, Nash equilibrium a Pareto optimální strategie?
- Najděte další Nash equilibrium ve smíšených strategiích.
- Ve smíšené strategii si každý hráč zvolí pravděpodobností rozložení možných strategií a chce maximalizovat očekávanou hodnotu užitku.

### Zadání (zkráceno)

#### Pravidla hry

- Máme dva hráče, kteří po jednom odebírají kameny očíslované  $1, 2, \dots, n$ .
- Hráč může odebrat libovolný kámen, který je násobek nebo dělitel předchozího odebraného čísla.
- Pokud hráč nemůže odebrat kámen, tak prohrává.

Napište funkci, která dostane seznam zbývajících kamenů a poslední odebraný kámen a rozhodne, zda je daná situace vyhrávající a pokud ano, tak vrátí kámen, který má hráč odebrat v dalším tahu.