Jozef Waldhauser

Úloha 5

Evolučne generovaná stratégia pre iterované väzňovo dilema

Obsah

[Úloha 5 – zadanie 2](#_Toc197254059)

[Úloha 5 – zdroje 2](#_Toc197254060)

[Teória 3](#_Toc197254061)

[Prečo práve evolučný algoritmus? 4](#_Toc197254062)

[Design algoritmu 4](#_Toc197254063)

[Overview 4](#_Toc197254064)

[Implementácia 6](#_Toc197254065)

[Implementácia reaktívnej tabuľky 6](#_Toc197254066)

[Implementácia funkcie zrada 7](#_Toc197254067)

# Úloha 5 – zadanie

1. navrhnout kódování tabulky reaktivního agenta, vracejícího další tah 0 (nezradit) či 1 (zradit) na základě historie vlastních tahů a tahů protihráče
2. evolučně najít 0/1 genom, který nejlépe hraje iterované vězňovo dilema
3. implementovat na základě vyevolvovaného genomu funkci zrada(moje\_hostorie, protihracova\_historie) vracejici nasledujici tah 0/1

**Úlohou bolo navrhnúť kódovanie tabuľky reaktívneho agenta, na základe histórie ťahov, evolučne nájsť 0/1 genóm, ktorý najlepšie hrá iterované väzňovo dilema a implementovať funkciu zrada(moje\_historie, protihracova\_historie) return 0 or 1.**

# Úloha 5 – zdroje

1 [What Game Theory Reveals About Life, The Universe, and Everything - Veritasium](https://www.youtube.com/watch?v=mScpHTIi-kM)

2 [The Iterated Prisoner's Dilemma and The Evolution of Cooperation](https://www.youtube.com/watch?v=BOvAbjfJ0x0)

3 [How to outsmart the Prisoner’s Dilemma - Lucas Husted](https://www.youtube.com/watch?v=emyi4z-O0ls)

4 [Axelrodov turnaj - Axelrod's tournament](https://cs.stanford.edu/people/eroberts/courses/soco/projects/1998-99/game-theory/axelrod.html)

5 [https://axelrod.readthedocs.io/ dokumentácia](https://axelrod.readthedocs.io/en/fix-documentation/reference/overview_of_strategies.html)

6 <https://www.sciencedirect.com/science/article/abs/pii/S0022519306004863>

7 <https://www.jessiefin.com/assets/pdf/tog-coop-ipd.pdf>

# Teória

Väzňovo dilema je tzv. „game theory challenge“, kde dvaja hráči vyberajú či protivníka zradia alebo nie. Podľa toho, ako sa rozhodnú dostanú obaja body.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  | Hráč 1 | |
|  |  | | Zradiť | Nezradiť |
|  | Zradiť | | 1, 1 | 5, 0 |
| Hráč 2 à | Nezradiť | | 0, 5 | 3, 3 |

Čiže ak sa obaja rozhodnú zradiť, tak dostanú každý jeden bod.

Ak jeden zradí a druhý nezradí tak dostanú 0 a 5 bodov – obeť 0 zradca 5.

Ak sa obaja nezradia tak dostanú 3 body.

V analýzach tohto problému [(zdroj 2, 3)](#_Úloha_5_–) sa často spomína že najlepšia možnosť je vždy zradiť, pretože keď si vyberieme nezradiť tak nedostaneme žiadne body. Skutočnosť je ale taká, že ak takto budú obaja hráči rozmýšľať tak vzájomná zrada im prinesie najmenší počet bodov.

Iterované väzňovo dilema túto situáciu dramaticky mení, keďže vidíme čo protihráč urobil ako posledný krok, tak môžeme podľa jeho rozhodnutí vedieť ako proti nemu „hrať“ resp. bojovať.

Je veľa stratégií, ako spomínané vo [videu 2](#_Úloha_5_–) tie teraz vymenujem a v krátkosti opíšem:

* **Random –** náhodne zradiť alebo nezradiť
  + nepredvídateľná
* **Always cooperate –** vždy nezradiť / spolupracovať
  + nevýhoda je v tom, že ak súper vždy zradí nebudeme mať body
* **Always defect –** vždy zradiť / nespolupracovať
  + ak budú obaja zrádzať nemaximalizujeme počet bodov
* **Tit-for-tat** (oko za oko) – začína sa spoluprácou a kopíruje posledný ťah
  + ak na začiatku nespolupracuje, môžeme mať celkovo menej bodov
  + môže sa dostať do cyklu spolupráca – nespolupráca (exploitable)
* **Tit-for-two-tats** – ak protihráč 2x zradí, tak až potom zradí
  + nevýhodné proti stratégií, ktorá skúša zradu
  + odolnejšie proti zacykleniu ako tit-for-tat
* **Forgiving tit-for-tat –** tit-for-tat ale má šancu odpustiť zradu
* **Grudger / Grim trigger –** spolupráca a ak protihráč raz zradí tak vždy zradí
  + nevýhodné proti väčšine stratégií
* **Pavlov –** ak predošlý ťah viedol k vysokej výplate, opakovať ho, ak nie tak zmeniť
  + môže sa zacykliť na zradách

## Prečo práve evolučný algoritmus?

**Evolučný algoritmus** dokáže lepšie vyriešiť opakovanú väzňovu dilemu, **lebo sa postupne prispôsobuje a učí**, zatiaľ čo **pevné stratégie zostávajú rovnaké**.  
Áno, niektoré stratégie sú zložitejšie a môžu byť úspešné, ale **evolúcia vie nájsť ešte lepšie riešenia tým, že ich kombinuje a zlepšuje.**

# Design algoritmu

Evolvovaný IPD bude bojovať proti 4 súperom (ktorí sa môžu zapnúť/vypnúť):

OPPONENTS = [

    ("AlwaysCooperate", always\_cooperate),

    ("AlwaysDefect", always\_defect),

    ("TitForTat", tit\_for\_tat),

    ("Random", random\_strategy)

]

**1. Inicializácia a nastavenie**

* **Nastavenie IPD a genetického algoritmu** – dĺžka histórie, počet kôl, veľkosť populácie, mutácie, kríženie atď.
* **Definícia IPD tabuľky odmien** – podľa pravidiel bežnej väzňovej dilemy.
* **Výber protihráčov**  – výber stratégií *(Always Cooperate, Always Defect, Tit for Tat, Random)* cez True/False booleans.
* **Načítanie knižnice DEAP**

**2. Vygenerovanie počiatočnej populácie**

* Genómy populácie sú náhodne vygenerované v dĺžke GENOME\_LENGTH.

**3. Fitness funkcia**

* Pre každý genóm sa spustia hry proti vybraným oponentom.
* Výsledné skóre (výhry) sa spriemerujú a použijú ako fitness hodnota.

**4. Dekódovanie stratégie**

* Genóm sa prekladá na rozhodovanie agenta podľa binárne zakódovanej histórie oboch hráčov.
* Základný stav je že spolupracuje.

**5. Simulácia hier (Play Game)**

* Počas NUM\_ROUNDS kôl hrá agent proti každému zvolenému oponentovi a zbiera body na základe tabuľky IPD.
* Agentove rozhodnutia sú určované genómom, oponenti používajú svoju (pevne danú) stratégiu.

**6. Genetický algoritmus (Evolúcia)**

* Beží počas MAX\_GENERATIONS generácií.
* Používa kríženie, mutácie, výber cez turnaje, a elitizmus.
* DEAP eaSimple funkcia iteruje populáciu a uchováva záznamy o výkonnosti.

**7. Vyhodnotenie a interpretácia výsledkov**

* Najlepší jedinec (genóm) sa vyberie a analyzuje.
* interpret\_strategy() vypíše rozhodovanie agenta pre každú možnú históriu.
* test\_funkcie\_zrada() simuluje rôzne historické situácie a testuje správanie finálnej stratégie.

**8. Vizualizácia**

* plot\_evolution() vykreslí priebeh fitness hodnôt v čase – max, priemer, min.

# Implementácia

## Implementácia reaktívnej tabuľky

|  |
| --- |
| def decode\_strategy(genome, agent\_history, opponent\_history):      if len(agent\_history) < HISTORY\_LENGTH or len(opponent\_history) < HISTORY\_LENGTH:          return 0  # ak nie je dostatok historie, spolupracujeme        recent\_agent\_moves = agent\_history[-HISTORY\_LENGTH:]      recent\_opponent\_moves = opponent\_history[-HISTORY\_LENGTH:]        genome\_index = 0        for i in range(HISTORY\_LENGTH):          genome\_index = genome\_index << 1          genome\_index = genome\_index | recent\_agent\_moves[i]          genome\_index = genome\_index << 1          genome\_index = genome\_index | recent\_opponent\_moves[i]      return genome[genome\_index] |

Pri návrhu *reaktívnej tabuľky agenta* som sa inšpiroval dokumentom [7 v zdrojoch](#_Úloha_5_–) (strana 2 v dokumente), kde autori používajú tzv. **L64 lookup table** ako reprezentáciu stratégie v IPD. Táto tabuľka má 64 binárnych hodnôt, pričom každá hodnota reprezentuje rozhodnutie agenta (spolupráca alebo zrada)

64 binárnych hodnôt bolo podľa mňa veľa tak som koncept prispôsobil - zmenšil históriu (konkrétne HISTORY\_LENGTH = 2), čo vedie na 4-bitový zápis predošlých pohybov (2 agentove + 2 súperove), teda celkovo 16 možných kombinácií.

Tento zápis využívam ako index do **genómu**, ktorý slúži ako **lookup table.**

Následne je genóm evolvovaný genetickým algoritmom. Rovnako ako v dokumente -každé rozhodnutie závisí od minulých krokov, čo robí agenta reaktívnym.

**Implementácia:**

* + Genóm je reprezentovaný ako zoznam hodnôt 0 a 1 (spolupráca alebo zrada)
  + Každý prvok v genóme predstavuje jedno rozhodnutie pre konkrétnu kombináciu histórie
  + Pre HISTORY\_LENGTH=2, existuje 2^4 = 16 možných kombinácií histórie (4 bity - 2 agentove a 2 akcie súpera)
  + Genom má preto dĺžku 16 (GENOME\_LENGTH = HISTORY\_LENGTH^4)

**Ako funguje mapovanie histórie na akciu:**

* + Vezmú sa posledné akcie agenta a súpera (podľa HISTORY\_LENGTH)
  + Z týchto akcií sa vytvorí binárne číslo (index do genomu)
  + Hodnota na tomto indexe v genome určuje ďalší ťah (0=spolupráca, 1=zrada)

### Príklad:

Ak je **HISTORY\_LENGTH=2** a história je:

Agentove posledné 2 ťahy: **[0,1] (spolupráca, zrada)**

Súperove posledné 2 ťahy: **[1,0] (zrada, spolupráca)**

Vytvorí sa index: 0110 binárne = 6 decimálne

Akcia bude: genome[6]

## Implementácia funkcie zrada

|  |
| --- |
| # globalna premenna pre ulozenie najlepsieho genomu  BEST\_GENOME = None  def zrada(moja\_historia, protihracova\_historia):      global BEST\_GENOME        if BEST\_GENOME is None:          return 0        # dependentna na decode\_strategy      return decode\_strategy(BEST\_GENOME, moja\_historia, protihracova\_historia) |

Funkciu *zrada* som implementoval využitím globálnej premennej BEST\_GENOME, tento genóm slúži ako **vstup pre funkciu decode\_strategy (reaktívna tabuľka),** ktorá následne vráti ďalší krok na základe genómu a histórie krokov hráčov.

**BEST\_GENOME** je získaný **z hall of fame** v main funkcii.