

# Evolučné programovanie\*

Matej Pakán

ID: 110867

Slovenská technická univerzita v Bratislave  
Fakulta informatiky a informačných technológií  
`xpakan@stuba.sk`

12. novembra 2021

---

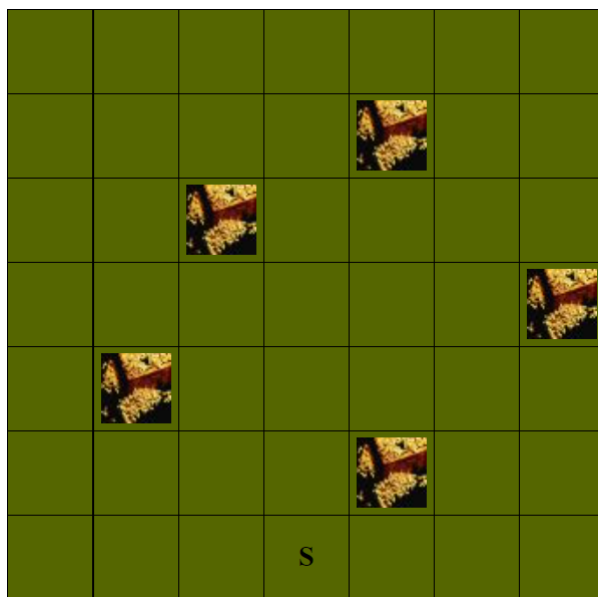
\*Riešenie 3. zadanie - Evolučné programovanie - hľadač pokladov – v predmete umelá inteligencia, ak. rok 2021/22, cvičiaci: Ing. Ivan Kapustík

## Obsah

<b>1</b>	<b>Zadanie</b>	<b>3</b>
<b>2</b>	<b>Úvodné nastavenia</b>	<b>3</b>
<b>3</b>	<b>Reprezentácia jedinca</b>	<b>4</b>
<b>4</b>	<b>Generovanie prvej generácie</b>	<b>4</b>
<b>5</b>	<b>Virtuálny stroj</b>	<b>5</b>
<b>6</b>	<b>Hodnotenie jedinca</b>	<b>5</b>
<b>7</b>	<b>Evolučný algoritmus</b>	<b>5</b>
7.1	Elitarizmus . . . . .	6
7.2	Výber rodičov a turnajový systém . . . . .	6
7.3	Mutovanie jedincov . . . . .	6
<b>8</b>	<b>Vývoj fitness</b>	<b>6</b>
<b>9</b>	<b>Testovanie</b>	<b>6</b>
9.1	Porovnanie parametrov a spôsobov selekcie . . . . .	7
<b>10</b>	<b>Možné vylepšenia</b>	<b>7</b>

## 1 Zadanie

Majme hľadača pokladov, ktorý sa pohybuje vo svete definovanom dvojrozmernou mriežkou (viď. obrázok) a zbiera poklady, ktoré nájde po ceste. Začína na políčku označenom písmenom S a môže sa pohybovať štyrmi rôznymi smermi: hore H, dole D, doprava P a doľava L. K dispozícii má konečný počet krokov. Jeho úlohou je nazbierať čo najviac pokladov. Za nájdenie pokladu sa považuje len pozícia, pri ktorej je hľadač aj poklad na tom istom políčku. Susedné políčka sa neberú do úvahy.



Obr. 1: Vzorové zadanie problému

Horeuvedenú úlohu riešite prostredníctvom evolučného programovania nad virtuálnym strojom.

Tento špecifický spôsob evolučného programovania využíva spoločnú pamäť pre údaje a inštrukcie. Pamäť je na začiatku vynulovaná a naplnená od prvej bunky inštrukciami. Za programom alebo od určeného miesta sú uložené inicializačné údaje (ak sú nejaké potrebné). Po inicializácii sa začne vykonávať program od prvej pamäťovej bunky. (Prvou je samozrejme bunka s adresou 000000.) Inštrukcie modifikujú pamäťové bunky, môžu realizovať vetvenie, programové skoky, čítať nejaké údaje zo vstupu a prípadne aj zapisovať na výstup. Program sa končí inštrukciou na zastavenie, po stanovenom počte krokov, pri chybnnej inštrukcii, po úplnom alebo nesprávnom výstupe. Kvalita programu sa ohodnotí na základe vyprodukovaného výstupu alebo, keď program nezapisuje na výstup, podľa výsledného stavu určených pamäťových buniek.

## 2 Úvodné nastavenia

Hneď nazačiatok som si vytvoril funkciu na čítanie zadania zo súboru. Tieto údaje načítavam po celých riadkoch. Na prvom riadku sa nachádza rozmer mrie-

žky (7x7), na druhom je štartová pozícia hľadača (x,y). Na zvyšných riadkoch až dokonca súboru načítavam súradnice pokladov (x,y).

Pre správnu funkčnosť som použil externé knižnice - random, time a math.

### 3 Reprezentácia jedinca

Každá **entita** (jedinec) obsahuje svoj **genóm** (zoznam génom), **fitness** (miera sily/úspešnosti jedinca) a **prints** (pohyb/výpis).

Object jedinca obsahuje aj funkciu vykonania pohybu po mriežke, k čomu sa dostaneme neskôr v dokumentácii.

### 4 Generovanie prvej generácie

Úvodnú generáciu generujem nasledovne:

Vytvorím si objekt **jedinca**. Tomuto jedincovi následne **vytváram 64 génov**.

- Tento gén, má následne na **prvých 2 bitoch zapísanú inštrukciu** tak, ako je napísané v zadaní (hodnoty od 00 po 11).
- **Ďalších 6 bitov reprezentuje adresu genómu** (číslo od 0 do 63) v binárnej sústave.
- Prvé 2 bity (inštrukcia) teda **rozhodujú** o tom, **aká inštrukcia sa vykoná nad ďalšími 6 bitmi** (adresa).
- Takýto gén **pridávam do celkového genómu** (zoznam) jedinca.
- Takto vytvoreného jedinca pridávam do 1. generácie.

Následne spúšťam funkciu virtuálneho stroja nad týmto jedincov a hneď potom daného jedinca ohodnotím a pridám mu tak fitness. - Viac o tejto časti je opísané v ďalšej sekcii.

Tento postup opakujem **100 krát** (počet jedincov v generácii).

Kód vytvorenia počiatočnej generácie:

```
1 | for count in range(N_IN_GENERATION):
2 |     # Generating entity
3 |     entity = Entity()
4 |     entity.genome = []
5 |     for i in range(64):
6 |         # Generating random gene from 0 to 255
7 |         gene_int = randint(0,255)
8 |         gene_bin = bin(gene_int)[2:].zfill(8)
9 |
10 |        # Adding genome to entity
11 |        entity.genome.append(gene_bin)
12 |
13 |        # Running VM on entity and getting fitness
14 |        VM(entity)
15 |        rateEntity(entity)
16 |
17 |        # Adding entity into generation
18 |        generation.append(entity)
```

## 5 Virtuálny stroj

Funkciu virtuálneho stroja potrebuje jedinca ako parameter.

Následne vykonáva akcie nad každým génom z genómu jedinca a tento gén modifikuje.

Ak sa na prvých 2 bitov genómu vyskytuje '00', vykoná inkrementácia adresy génu. To znamená, že ak bolo v géne zapísané '001011', tak sa tento gén prepíše na '001100'.

Pri bitoch '01' - inštrukcia dekrementácie, funguje prepisovanie analogicky, akurát opačným smerom, teda odpočítavame.

Zaujímavá je funkcia JUMP ('10'), v pri tejto funkcii daný gén ignorujeme skáčeme na gén, ktorý je zapísaný v adresovej časti JUMP gény.

Poslednou funkciou je výpis ('11'), ktorá vykonáva funkciu addDirection v objekte jedinca, teda mu pridáva ďalší krok, ktorý hľadač vykoná. Smer pohybu jedinca je určený poslednými 2 bitmi génu.

```

1 |         def addDirection(self, bits):
2 |             direction = ""
3 |             if(bits == '00'):
4 |                 direction = "H"
5 |             elif(bits == '01'):
6 |                 direction = "D"
7 |             elif(bits == '10'):
8 |                 direction = "P"
9 |             elif(bits == '11'):
10 |                 direction = "L"
11 |             self.prints.append(direction)

```

Ak by sa mal tento virtuálny stroj zacykliť, máme na to podmienku, že ak stroj spraví viac ako 500 inštrukcií, virtuálny stroj skončí.

## 6 Hodnotenie jedinca

Na hodnotenie jedinca slúži funkcia rateEntity. Táto funkcia zoberie už existujúci zoznam výpisov/pohybov jedinca, ktorý predtým vygeneroval virtuálny stroj.

Následne prechádza každý z pohybov a zisťuje, či je tento pohyb možný (teda či náhodou hľadač nevybočil z mriežky). Ak je táto podmienka splnená, vykoná sa pohyb v danom smere. Po vykonaní tohto pohybu sledujeme, či sa daný jedinec nenachádza na mieste kde je poklad. Ak áno, tento poklad vyhlasujeme za nájdený aby ho hľadač nehľadal znovu a zvyšujeme mu jeho fitness o 1 bod. Pokiaľ hľadač nenašiel všetky poklady, resetujeme zoznam pokladov na pôvodné súradnice. Fitness ale už ostáva uložený.

## 7 Evolučný algoritmus

Pri generovaní každej ďalšej generácie využívam rovnaký postup.

- Nakopírujem najlepších X jedincov do novej generácie
- Krížim Y nádejných jedincov z predošlej generácie a pridávam ich do novej generácie

- Vytvorím Z úplne nových, náhodných jedincov
- Nad každým jedincom z generácie spúšťam virtuálny stroj
- Hodnotím každého jedinca
- Ak som nenašiel všetky poklady, opakujem tento postup

## 7.1 Elitarizmus

Pri mojom programe využívam elitarizmus, nakoľko som s ním dosiahol optimálnejšie výsledky. Elitarizmus spočíva iba v tom, že nakopíruje najlepších jedincov zo starej generácie do novej generácie bezo zmeny.

## 7.2 Výber rodičov a turnajový systém

Na začiatku kríženia jedincov si vyberiem rodičov. To je N najlepších jedincov z predošlej generácie. Následne podľa konfigurácie v úvode programu doplním počet jedincov krížením.

Výber rodičov prebieha turnajovým systémom, t.j. vyberiem 4 náhodným rodičov a vytvorím "pavúkový" eliminačný systém turnaja. V každom súboji nádejných rodičov sa porovnáva fitness - ten lepší postupuje do ďalšieho kola turnaja. Víťaz tohto turnaja sa následne stáva rodičom.

Tento turnaj opakujeme ešte druhý krát, keďže rodičia musia byť dvaja.

Následne týchto jedincov krížim. Samotné kríženie spočíva v tom, že 64 krát doplním gén do genómu nového jedinca, tak že náhodne (50 na 50) doplním N-tý 1. rodiča alebo 2. rodiča. Keď toto opakovanie skončí, mám vytvoreného nového jedinca s celým genómom.

## 7.3 Mutovanie jedincov

Jedinec, ktorý bol vygenerovaný krížením má šancu mutovať (2% alebo podľa konfigurácie pri každom géne). Znamená to, že prechádzam genóm jedinca (64 génov) a pri každom géne je šanca, že daný gén kompletne zmutuje - vytvorí sa nový gén.

# 8 Vývoj fitness

Hodnota fitness u mňa spočíva iba v počte nájdených truhlíc. Mám implementované vylepšovanie fitness aj na základe počtu krokov, ale výsledky mám pri použití počtu krokov horšie (z mne neznámeho dôvodu :( ).

# 9 Testovanie

Testovanie som realizoval na notebooku s procesorom i5-7300HQ pri frekvencii procesora 3.2 - 3.5GHz.

Ako môžete vidieť na testovaní vzorového riešenia, výsledky neboli úplne konštantné. Správne riešenia som našiel v 7 z 10 prípadoch, ak som generoval menej ako 10 000 generácií. Najrýchlejšie mi algoritmus pracoval týchto nastaveniach:

- Miera mutácie - 2%
- Elitarizmus - 20 jedincov
- Výber rodičov pozostával zo 60 najlepších jedincov
- V každej novej generácii som generoval 10 úplne nových jedincov
- Turnajový systém bol využitý

Pri výpise riešenia uvádzam pomocou akých krokov sa k riešeniu podarilo dostať pomocou vypísania jednotlivých krokov v zozname.

### 9.1 Porovnanie parametrov a spôsobov selekcie

Výsledky testovania môžete nájsť na konci tohto dokumentu.

## 10 Možné vylepšenia

Určite by pomohlo, ak by sa mi podarilo lepšie implementovať zlepšovanie fitness na základe vykonaného počtu krokov jedinca.