Evolučné programovanie*

Matej Pakán
ID: 110867
Slovenská technická univerzita v Bratislave
Fakulta informatiky a informačných technológií
xpakan@stuba.sk

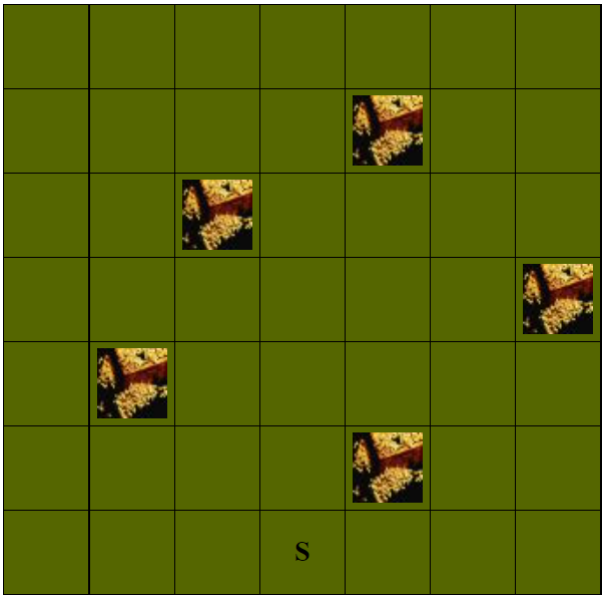
12. novembra 2021

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

Obsah

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 riešte 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 chybnej 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 súbora 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).

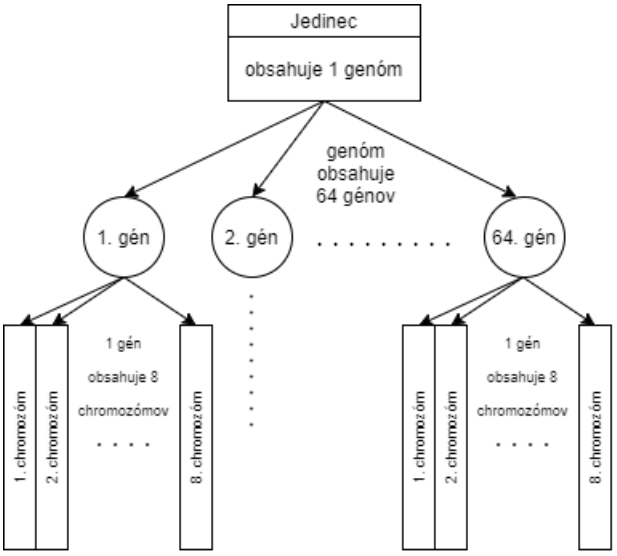
Objekt jedinca obsahuje aj funkciu vykonania pohybu po mriežke, k čomu sa dostaneme neskôr v dokumentácii.

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 vykonú nad ďalšími 6 bitmi** (adresa).
- Takýto gén **pridávam do celkového genómu** (zoznam) jedinca.
- Takto vytvoreného jedinca pridávam do 1. generácie.



Obr. 2: Zloženie jedinca

Následne spúšťam funkciu virtuálneho stroja nad týmto jedincom 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:

```
for count in range(N_IN_GENERATION):
    # Generating entity
    entity = Entity()
    entity.genome = []
    for i in range(64):
        # Generating random gene from 0 to 255
        gene_int = randint(0,255)
        gene_bin = bin(gene_int)[2:].zfill(8)

        # Adding genome to entity
        entity.genome.append(gene_bin)

    # Running VM on entity and getting fitness
    VM(entity)
    rateEntity(entity)

    # Adding entity into generation
    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 sa **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.

Zaujímava je funkcia **JUMP** ('10'), v pri tejto funkcii daný gén ignorujeme a skáčeme na gén, ktorý je zapísaný v adresovej časti JUMP génu.

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**.

```
    def addDirection(self,bits):
        direction = ""
        if(bits == '00'):
            direction = "H"
        elif(bits == '01'):
            direction = "D"
        elif(bits == '10'):
            direction = "P"
        elif(bits == '11'):
            direction = "L"
        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.

1. Nakopírujem najlepších X jedincov do novej generácie
2. Krížim Y nádejných jedincov z predošlej generácie a pridávam ich do novej generácie
3. Vytvorím Z úplne nových, náhodných jedincov
4. Nad každým jedincom z generácie spúšťam virtuálny stroj
5. Hodnotím každého jedinca
6. 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 dopĺnam počet jedincov krížením.

Výber rodičov prebieha turnajovým systémom, t.j.vyberiem 4 náhodným rodičov a vytvorím "pavúkovýëliminačný 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, kedže 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 genónom.

Obr. 3: Zloženie generácie

7.3 Mutovanie jedincov

Jedinec, ktorý bol vygenerovaný krížením má šancu mutovať (2% alebo podľa konfigurácie pri každom géne). Znamená to, že prechádzam genónom jedinca (64 génov) a pri každom genóme je šanca, že daní 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.

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

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

9.1 Porovnanie parametrov a spôsobov selekcie

Výsledky testov môžete nájsť na konci tohto dokumentu.

	Vzorové zadanie	
Počet jedincov v generácii	100	
Max. počet generácii	10000	
Miera mutácie (%)	2	
Elitarizmus	20	
Počet rodičov	60	
Generovanie náhodných detí	0	
Turnajový systém	NIE	
Test	**ČAS**	**GENERÁCIA RIEŠENIA**
1	8,83	978
2	6,98	773
3	11,02	1162
4	11,49	1221
5	65,34	7589
6	9,94	1091
7	49,17	5543
8	NEDOKONČIL	
9	NEDOKONČIL	
10	NEDOKONČIL	
Vyriešených	**70%**	
Priemerný čas vyriešených (s)	**23,25**	
Priemerná konečná generácia	**2622**	

Obr. 4: Výsledky testovania bez turnajového typu selekciu a bez náhodného generovania jedincov.

10 Možné vylepšenia

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

	Vzorové zadanie	
Počet jedincov v generácii	100	
Max. počet generácii	10000	
Miera mutácie (%)	2	
Elitarizmus	20	
Počet rodičov	60	
Generovanie náhodných detí	10	
Turnajový systém	ÁNO	
Test	**ČAS**	**GENERÁCIA RIEŠENIA**
1	6,16	537
2	2,96	260
3	21,38	1981
4	31,00	2810
5	27,10	2421
6	25,31	2339
7	3,55	309
8	NEDOKONČIL	
9	NEDOKONČIL	
10	NEDOKONČIL	
Vyriešených	**70%**	
Priemerný čas vyriešených (s)	**16,78**	
Priemerná konečná generácia	**1522**	

Obr. 5: Najlepšie výsledky - Použitie turnajového typu selekcie a generovanie náhodných jedincov. Vyššia miera elitarizmu (5%)

	Vzorové zadanie	
Počet jedincov v generácii	100	
Max. počet generácii	10000	
Miera mutácie (%)	2	
Elitarizmus	10	
Počet rodičov	60	
Generovanie náhodných detí	10	
Turnajový systém	ÁNO	
Test	**ČAS**	**GENERÁCIA RIEŠENIA**
1	6,01	454
2	10,88	853
3	33,84	2579
4	51,91	4060
5	14.57	1156
6	7,22	563
7	NEDOKONČIL	
8	NEDOKONČIL	
9	NEDOKONČIL	
10	NEDOKONČIL	
Vyriešených	**60%**	
Priemerný čas vyriešených (s)	**21,97**	
Priemerná konečná generácia	**1611**	

Obr. 6: Použitie turnajového typu selekcie a generovanie náhodných jedincov. Nižšia miera elitarizmu (10%)

	Vzorové zadanie	
Počet jedincov v generácii	100	
Max. počet generácii	10000	
Miera mutácie (%)	2	
Elitarizmus	5	
Počet rodičov	60	
Generovanie náhodných detí	10	
Turnajový systém	ÁNO	
Test	**ČAS**	**GENERÁCIA RIEŠENIA**
1	83,18	1287
2	4,77	338
3	31,37	1311
4	147,46	4978
5	11,57	842
6	9,64	599
7	6,20	448
8	NEDOKONČIL	
9	NEDOKONČIL	
10	NEDOKONČIL	
Vyriešených	**70%**	
Priemerný čas vyriešených (s)	**42,03**	
Priemerná konečná generácia	**1400**	

Obr. 7: Použitie turnajového typu selekcie a generovanie náhodných jedincov. Nižšia miera elitarizmu (5%)