Slovenská technická univerzita

Fakulta informatiky a informačných technológií

Ilkovičova 3, 842 16 Bratislava 4

Umelá Inteligencia

Zadanie 3

Hľadanie pokladu

Matsveyeva Lada-Ivanna

ZS 2021/22

Obsah:

Hľadanie pokladu 1
Úloha 3
Reprezentácia údajov 4
Opis algoritmu 4
Testovanie a porovnanie výsledkov 6
Zhrnutie a zhodnotenie riešenia 7

Matsveyeva Lada-Ivanna, ID: 99198

Úloha

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.

Zadanie

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.

Matsveyeva Lada-Ivanna, ID: 99198

Reprezentácia údajov:

```
class Generation:

    def __init__(self):
        self.individuals = []

    def add_individual(self, indiv):
        self.individuals.append(indiv)
```

Generácia je reprezentovaná triedou *Generation*. Každá generácia obsahuje zoznam jedincov. Trieda slúži len na lepšiu reprezentáciu dát.

```
class Individual:
    def __init__(self):
        self.fitness_value = 0
        self.memory_cells = []
        self.path = []
        self.found_treasures = []
        self.path_final = []
```

Pre každého jedinca:

fitness_value – fitness hodnota

memory_cells – pole, ktoré obsahuje pamäťové bunky

path – pole, ktoré obsahuje prvotný pohyb jedinca

found_treasures – nájdené poklady

path_final – finálna cesta jedinca

Opis algoritmu:

1. Zo vstupného súboru sa načíta rozmer mriežky, štartovacia pozícia, rozmiestnenie pokladov.

```
7 7
7 4
2 5
3 3
5 2
6 5
```

Vstupný súbor:

1 riadok – rozmer mriežky
2 riadok – štartovacia pozícia
3-6 riadky – rozmiestnenie pokladov

Pre spracovanie vstupného súboru slúži funkcia *load_map()*.

Inicializácia prvej generácie.
Polovica pamäťových buniek sa naplní náhodnými hodnotami od 0 po 255, keďže 1 bunka je 1 byte.
Počet jedincov je reprezentovaný *individual_count* s hodnotou 100. Podľa voľby používateľa sa dá zmeniť.

2. Spracovanie programu virtuálnym strojom, určenie fitness hodnoty.

   Funkcia *virtual_machine()*.

   Virtuálny stroj podľa zadania obsahuje 4 inštrukcie: inkrementácia, dekrementácia, skok na adresu, výpis. Prvé dva bity reprezentujú operáciu, ďalšie 6 je adresa pamäťovej bunky. Hodnota výpisu je určená podľa posledných dvoch bitov. "10" – L – doľava, "11" – R – doprava, "00" – U – hore, "01" – D – dole.

inštrukcia	tvar
inkrementácia	00XXXXXX
dekrementácia	01XXXXXX
skok	10XXXXXX
výpis	11XXXXXX

   Na začiatku skopíruje pamäťové bunky, kvôli tomu, že za behu programu sa budú modifikovať.

   Program sa skončí vykonávať vtedy ak presiahne hranicu pamäťových buniek alebo počet inštrukcií presiahne 500.

   V cykle postupne spracujeme každého jedinca. Po tom ako stroj skončil, program sa určí výslednú cestu na základe toho či je jedinec nepresiahol mapu. Tiež sa zistí, či je na danej bunke poklad. Ak nájde všetky poklady hľadanie cesty sa skončí.

   Ďalej sa určí fitness hodnota vo funkcie *fitness()*.

   Ak finálna cesta jedinca je prázdna, tak fitness hodnota je 0. Ak nie, tak fitness hodnota sa skladá s toho, koľko pokladov našiel jedinec a koľko krokov vykonal. Jedinec s menším počtom krokov má väčšiu fitness hodnotu. Výpočet fitness hodnoty vyzerá takto:

   fitness_hodnota = počet nájdených pokladov + (1 – (počet krokov * 0.001))

3. Ak jedinec našiel všetky poklady, program vypíše údaje. Ak nie tak vytvorí sa nová generácia.

   Vytvorenie novej generácie.

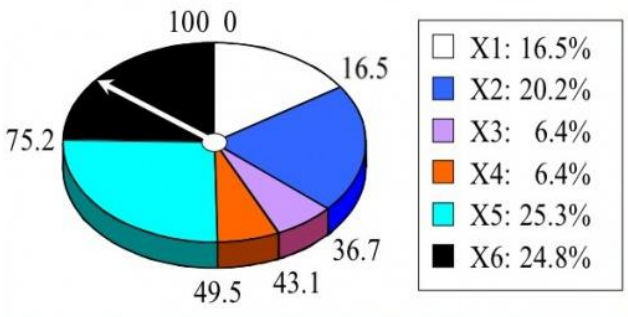
   Funkcia *create_new_generation()*.

   Časť novej generácie bude sa skladať s jedincov predošlej generácie, selekcia jedincov sa určí na základe metódy ruleta a turnaj. Dve metódy sa nepoužívajú spoločne, sú vytvorené na porovnanie výsledkov z rôznych typov selekcie.

   Metóda ruleta:

   Výber sa robí na základe fitness hodnoty. Čím má jedinec lepšiu hodnotu, tým dostáva väčšiu šancu.

   Na kolese rulety má každý jedinec vyhradený segment úmerný jeho fitness hodnote. Kedy sa vygenerované číslo zastaví na nejakom segmente jedinca, tento jedinec pôjde do novej generácie.



Metóda turnaj:

Z celej generácie sa vyberie náhodne dve skupiny jedincov, v každej skupine je práve dva jedinca. V každej skupine zostáne len jeden lepší jedinec, a práve tieto pôjdu do novej generácie.

Ostatné jedenci pre novú generáciu budú vytvorené pomocou kríženia a mutácie dvoch jedincov.

Kríženie

Funkcia *crossover()*.

Náhodne sa určí bod kríženia. Nové pamäťové bunky sa budú vytvorené tak, že do bodu kríženia budú bunky od prvého rodiča, ostatné po bode kríženia.

Mutácia

Funkcia *mutation()*.

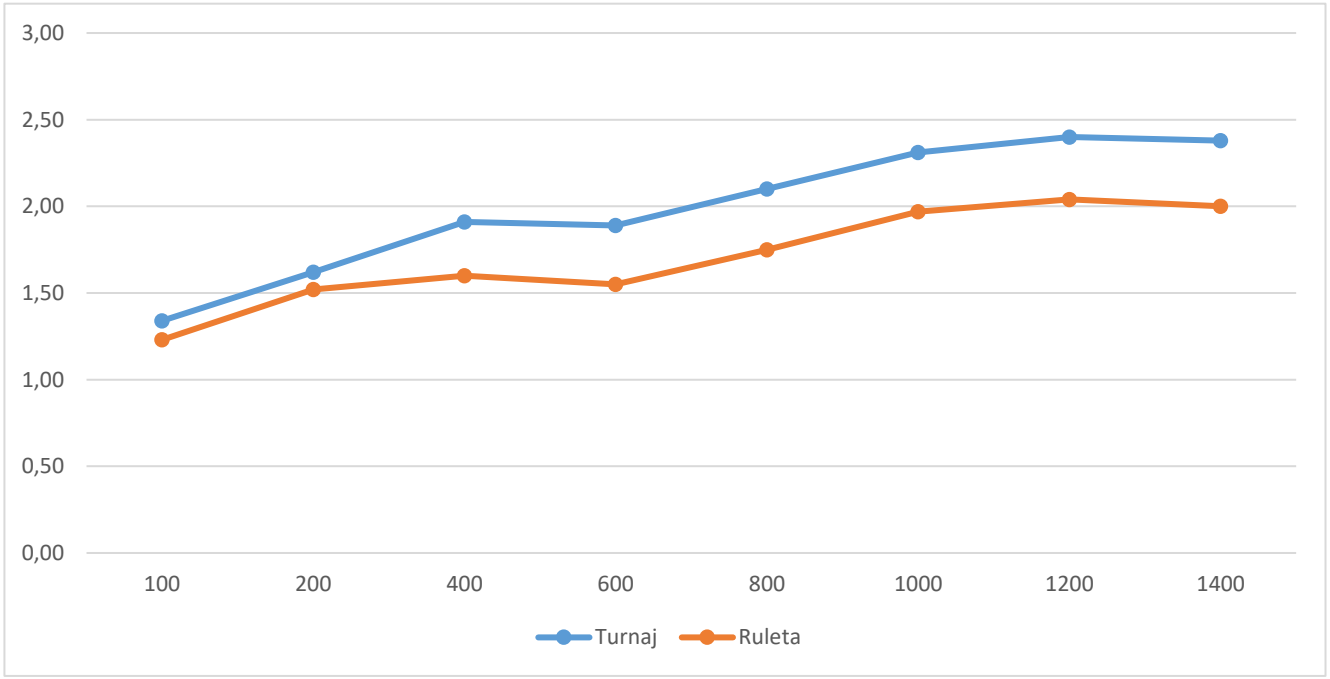
Vtedy keď percento mutovania je príliš veľké, hľadanie riešenia sa blíži náhodnému. Preto použitá na generovanie šanca – náhodne číslo od 0 po 1. Ak je šanca menšia ako 0.4 pamäťové bunky sa mutujú. Vyberie sa náhodná bunka a zmení sa na náhodne číslo od 0 do 255.

Teraz je vytvorená nová generácia, program prejde na bod 3.

Testovanie a porovnanie výsledkov

Vývoj fitness hodnoty.

Pri testovaní boli použité priemerné fitness hodnoty jedincov, nie je najlepšieho, kvôli tomu, že hodnota v generácie ktorá našla všetky poklady je: počet pokladov + 0 až 1. Pri testovaní bolo 7 pokladov.

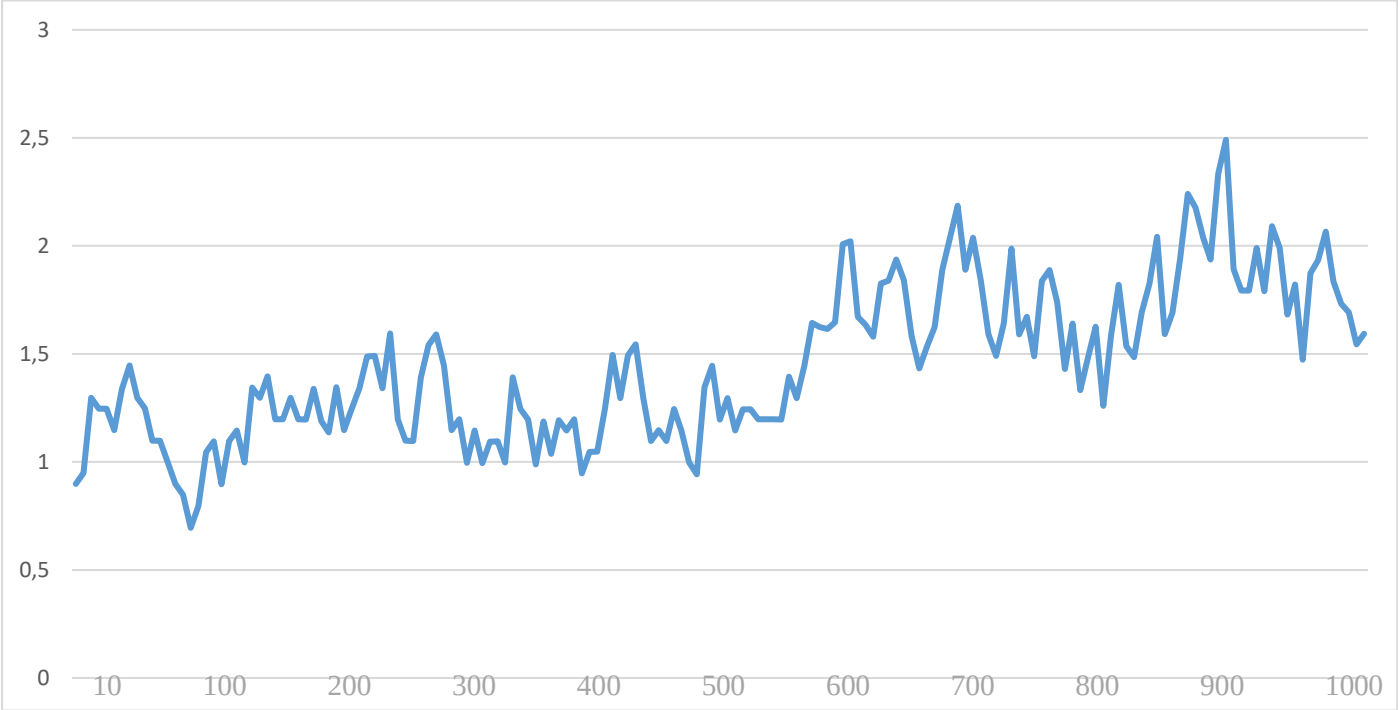


Podľa výsledkov testovania je vidieť, že selekcia metodom turnaj je efektívnejšia. Je to sposobené tým, že pri metóde turnaj vždy je možnosť zvoliť lepšieho v páre, keď je v metóde ruleta vyberie práve ten, na ktorý segment ukázalo. Výsledky testovania sa líšia viac len počtom spravených krokov, a nie je počtom nájdených pokladov.

Na začiatku fitness hodnota je rastúca, je to spôsobené tým, že prvé generácie nemajú veľa nájdených pokladov a tým pádom program sa skončí skôr, ako nájde všetky poklady, pri väčších počtoch generácie je fitness hodnota takmer rovnaká. Väščinou pri počte generácií viac ako 700, program skončí skôr, ako vygeneruje stanovený počet generácií.

Vývoj fitness hodnoty pri veľkom počte selekcie.

Fitness hodnota sa pohybuje od 0.7 až po 2.5, vo väčších prípadoch všetky poklady neboli nájdené, k tomu priviedlo to že sa generácie takmer nezmenili počas vývoja.



Zhrnutie a zhodnotenie riešenia

Na dosiahnutie lepších výsledkov je potrebné nastaviť určité hodnoty počtu jedincov v generacie, percentá selekcie a mutácie. Toto je možné dosiahnuť testovaním.

V tomto algoritme veľa krát hrá rolu náhodnosť, napríklad, pri zaplnení pamäťových buniek pre prvú generáciu, selekciu, kríženie. Toto spôsobuje tomu, že pri rovnakom rozmiestnení pokladov, štartovacej pozícii, počtu generácií a jedincov je možné dostať rôzne výsledky. Vo väčšine prípadov riešenie bolo nájdené v rozsahu od 700 po 1200 generácií. Percentá jedincov, ktoré vstúpia do novej generácie nemá byť príliš vysoké, lebo rýchle sa dostane k jednému genotypu v generácie. Taktiež percentá mutácie nemali by mať veľké hodnoty, aby nedošlo k slepému hľadaniu.