Slovenská technická univerzita

Fakulta informatiky a informačných technológií

Ilkovičova 3, 842 19 Bratislava

Ema Richnáková

Evolučný algoritmus

Zenová záhrada

Predmet: Umelá inteligencia

Akademický rok: 2019/2020

1. Úvod

V evolučnom algoritme sa dvaja rodičia vyberajú pomocou turnamentu a kríženie rodičov prebieha pomocou metódy dvojbodového kríženia. Pri krížení môže nastať aj mutácia. Elitarizmus sa tu deje v takom zmysle, že jedinec s najlepším fitnes sa dostane do ďalšej generácie.

2. Opis riešenia

2.1. Implementácia

Zdrojový kód programu je napísaný v programovacom jazyku Python.

2.2. Reprezentácia údajov v kóde

Údaje záhrady v kóde, ktoré ovplyvňujú veľkosť riešeného problému:

- garden_size = [x, y]
  - x predstavuje šírku záhrady, y predstavuje výšku záhrady
  - tieto údaje sa získajú z inicializačného súboru *zen_garden_parameters.txt*. Ak tento súbor nebol nájdený alebo ho program nevedel prečítať, vytvorí sa mapa záhrady podľa príkladu v zadaní.
- stones_num
  - premenná predstavuje počet kameňov v záhrade
  - v inicializačnom súbore je to ako 3 číslo v poradí. Ak nebolo uvedené, vyberie sa počet náhodne z rozmedzia od 1 po ¼ obsahu záhrady.
- print_garden_bool
  - premenná, ktorá je uvedená ako boolean hodnota a rozhoduje, či sa záhrady budú tlačiť do terminálu alebo nie. Ak nebola hodnota zadaná, záhrady sa nebudú tlačiť do terminálu.

2.3. Riešenie

Trieda *Evolution_algorithm* predstavuje súbor funkcií, ktoré vykonávajú evolučný algoritmus. Trieda má ako vstupný údaj záhradu so všetkými jej parametrami. Taktiež sú v nej nastavené konštanty, ktoré sú dôležité pri krížení či mutácií. Konštanta *POPULATION_SIZE* predstavuje počet jedincov v jednej populácii a je nastavená na 40 jedincov. Konštanta *CROSSOVER_RATE* udáva šancu na kríženie, ktorá je nastavená na 95%. Konštanta *GENERATIONS_MAX* si ukladá maximálny počet generácií, aby algoritmus pri nenájdenom konečnom riešení nešiel donekonečna. Maximálne môže zbehnúť 2500 generácií. Konštanty *MUTATION_MIN_PERC* a *MUTATION_MAX_PERC* udávajú rozmedzie pravdepodobnosti mutácie génu, ktorých hodnoty sú v rozmedzí od 0,05 až po 0,4. Mutácia nastáva len vtedy, ak sa jedinci krížia. Chromozóm jedinca má veľkosť rovnej *šírke záhrady + dĺžke záhrady + počet kameňov.* Pre prvú generáciu sa generujú chromozómy pomocou funckie *generate_chromosome()*

```python
# vygeneruje 1 chromozon
def generate_chromosome(self) -> list:
    chromosome = [] # list genov = chromozom
    perimeter_numbers = [] # cisla z obvodu zahrady (od 1 po 2*(sirka + vyska))

    for i in range(1, ((self.garden.half_perimeter) * 2) + 1):
        perimeter_numbers.append(i)

    random.shuffle(perimeter_numbers) # nahodne rozhodi cisla celeho obvodu

    for i in range(self.genome_num): # prvych n (= pocet genomov) vlozi do chromozonu
        num = perimeter_numbers[i]
        if random.randrange(0, 10) < 5:
            # sanca 50%, ze vygeneruje bud kladne alebo zaporne cislo -> to ovplyvni vyber smeru pri zrazke s prekazkou
            num *= -1

        chromosome.append(num)

    return chromosome
```

Keďže jedinec môže vchádzať zo všetkých strán záhrady, tak sa uložia do poľa čísla od 1 po hodnotu obvodu záhrady. Tieto čísla sa náhodne v poli usporiadajú a prvých *n* (n predstavuje počet génov v chromozóme) sa vyberie z poľa a uložia sa do chromozómu, ktorý je z funkcie vrátený.

Po vytvorení prvej generácie sa môže táto generácia pustiť do hrabania.

Funkcia hrabania je uložená v triede *Zen_Garden* vo funkcií *rake_graden()*

```python
# pohrabanie zahrady; chromosome = predstavuje chromozom mnicha, ktory predstavuje vstupne body do zahrady
# mnich moze vstupovat zo vsetkych stran zahrady
def rake_garden(self, chromosome : list) -> int:
    coordinate = {} # suradnice [vyska, sirka]
    new_coordinate = {} # nove suradnice [vyska, sirka]
    prev_coordinate = {} # predosle suradnice [vyska, sirka]
    crossing_num = 1 # cislo prechodu mnicha
    crossing_list = [] # list prechodov mnicha
    monks_garden = self.copy_garden() # skopiruje nepohrabanu zahradu pre kazdeho mnicha zvlast
    stuck_monk = False # uviaznutie mnicha

    for i in range(len(chromosome)): # mnich vstupuje do zahrady
        coordinate = self.get_direction(chromosome[i]) # podla cisla vstupu do zahrady vrati presne pole v zahrade

        if monks_garden[coordinate['row']][coordinate['col']] == self.SAND: # je volny vstup do zahrady?
            while self.is_field_in_garden(coordinate['row'], coordinate['col']): # pokial sa nachadza policko v zahrade, moze hrabat
                if monks_garden[coordinate['row']][coordinate['col']] == self.SAND: # ak je policko piesok
                    monks_garden[coordinate['row']][coordinate['col']] = crossing_num # zaznaci prechod cez policko
                    new_coordinate = {'row': coordinate['row'] + coordinate['move'][0], 'col': coordinate['col'] + coordinate['move'][1],
                                      'move': coordinate['move']}
                    prev_coordinate = coordinate
                    coordinate = new_coordinate

                else: # ak narazi (cize objavi sa na policku s prekazkou)
                    coordinate = self.solve_colission(chromosome[i], monks_garden, prev_coordinate) # zisti, kam sa ma pohnut podla genu
                    if coordinate['move'] == None: # ak nema moznost pohybu nikam
                        if not self.is_field_in_garden(coordinate['row'], coordinate['col']): # ak sa zasekol pri kraji zahrady
                            break # uspesne konci
                        else: # ak narazil na dalsiu prekazku
                            stuck_monk = True
                            break # neuspesne konci

                if stuck_monk:
                    break

                crossing_num += 1

    if self.print_garden_bool:
        self.print_garden(monks_garden) # vytlaci zahradu, ak je to povolene

    return self.count_raked_fields(monks_garden)
```

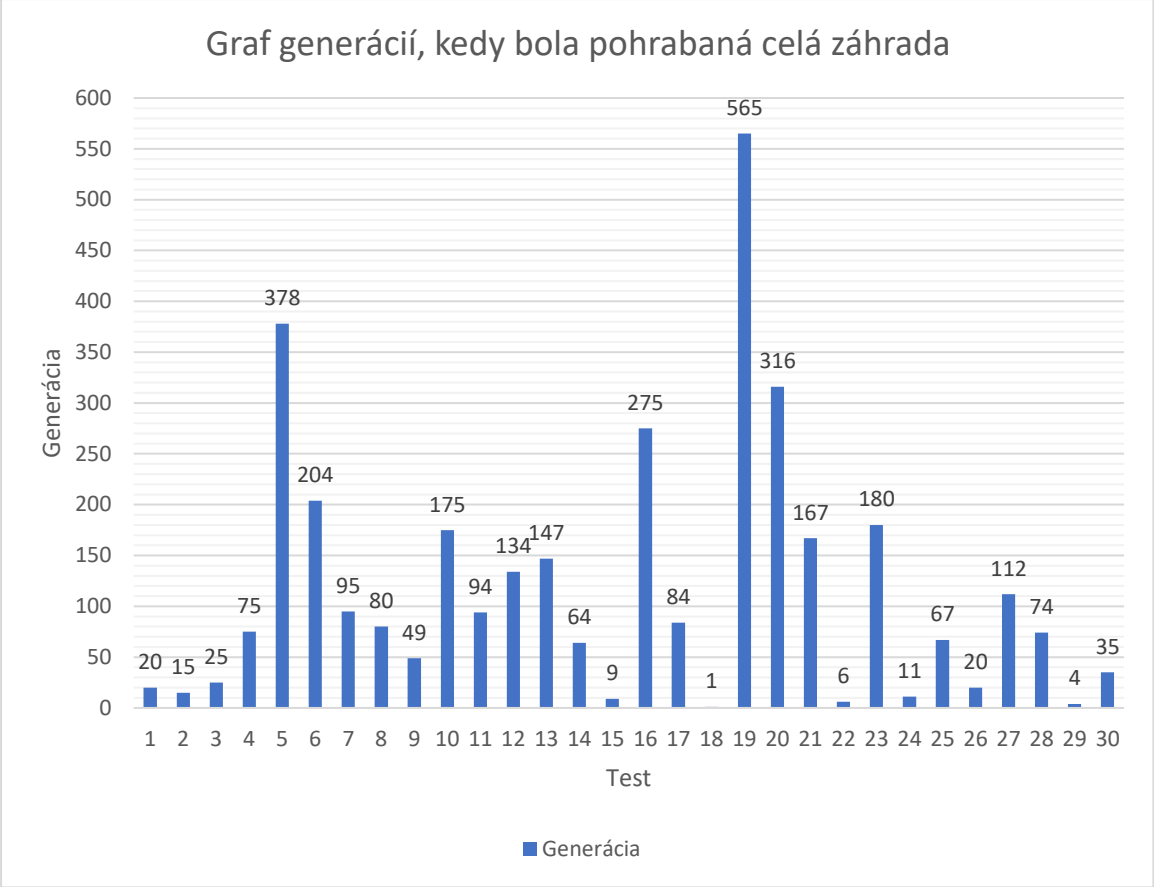
Ako vstupný údaj má chromozóm jedinca. Funkcia potom prechádza každý gén v chromozóme, aby zistila, akým políčkom vstupuje do záhrady. To zisťuje konkrétne funkciou *get_direction()*, kde sa vloží gén (teda číslo z obvodu záhrady) a z toho génu zistí, z akej strany vstupuje a vracia štruktúru typu dictionary, ktorá sa ukladá do premennej *coordinate*. Táto štrukúra v premennej má tvar *{'row': key1, 'col': key2, 'move': [a, b]}*, kde

v *row* je uložené číslo riadku, v *col* je uložené číslo stĺpca a v *move* sú uložené hodnoty -1, 0, 1, ktoré predstavujú pohyb dole ([1, 0]), hore ([-1, 0]), doprava ([0, 1]) alebo doľava ([0, -1]). Potom sa zisťuje, či je vstup do záhrady voľný (teda, či vstupuje na nepohrabaný piesok) a ak je vstup voľný, tak sa posúva buď vodorovne alebo zvislo vo while loop-e, označuje prejdené políčka číslom prechodu a stále kontroluje, či pri novom pohybe nevystúpil zo záhrady alebo nenarazil na prekážku. Ak vystúpil zo záhrady, zvýši číslo prechodu a číta ďalší gén, podľa ktorého znovu zisťuje vstupné políčko. Ale ak pri prechode jedinec narazí na prekážku, čiže buď kameň alebo pohrabané políčko, vtedy musí riešiť kolíziu, ktorá sa prepočítava vo funkcií *solve_colision().* Tá spočíva v tom, že ak sa doteraz mních pohyboval medzi stĺpcami, teda vodorovne, tak bude zahýbať buď hore alebo dole. To závisí od toho, či gén je kladný alebo záporný. Ak je kladný pôjde hore, ak záporný dole. Ak ale nebude mať zvolenú cestu voľnú, tak sa pozrie, či na opačný smer má cestu voľnú. Ak ani vtedy neuspeje, tak sa mních zasekol a už ďalej nehrabe. To isté platí, ak sa mních pohyboval medzi riadkami, teda zvislo a ak mal kladný gén, pozrie sa najprv doprava, ak záporný, tak doľava, ak sa zasekne, taktiež končí. Ale ak je mních zaseknutý, tak nelikvidujeme jeho chromozóm, ale má menšiu šancu vyhrať v turnaji pri výbere rodičov. Ak mních dohrabe záhradu, tak sa vráti z funkcie počet pohrabaných políčok, čo zároveň predstavuje fitnes daného jedinca. Tieto finesy sa ukladajú do poľa, ak celá jedna generácia dohrabala záhrady, zistí sa maximálna fitnes generácie a ak bola dosiahnutá optimálna fitnes, čiže boli pohrabané všetky políčka, tak evolučný algoritmus končí. Ak ani jeden jedinec nedosiahol optimálneho maximálneho fitnes a ešte sa nedosiahol maximálny počet generácií, tak sa vygeneruje ďalšia generácia vo funkcii *generate_new_population()*.

V tejto funkcii sa najprv vyberú dvaja rodičia pomocou turnaju. Potom ak nastane pravdepodobnosť 0,95, budú sa vybratí dvaja rodičia krížiť dvojbodovým krížením, z ktorého vzniknú dvaja noví potomkovia, ktorí majú ešte malú pravdepodobnosť na mutáciu. Tá prebieha tak, že sa vygeneruje náhodné číslo, ktoré sa môže nachádzať v chromozóme, potom je 50% šanca, že sa zmení znamienko tohoto čísla a potom sa hľadá, či sa presne také isté číslo nachádza v chromozóme. Ak nie, vloží nový gén namiesto aktuálneho génu, a ak sa nachádza duplikát v chromozóme, tak vymení aktuálny gén s duplikátom, resp. novo generovaným génom. Ak sa rodičia nekrížia, tak sa prekopírujú do novej generácie. Pomedzi vytváraním nových generácií, je počítadlo, ktoré kontroluje, ako dlho sa udržuje lokálne maximum. Ak 40 generácii je to isté maximum, tak sa prvá polovica populácie zachová a druhá sa nahradí novo vygenerovanou polovicou populácie.

3. Testovanie

Pri testovaní môjho evolučného algoritmu na príklade zo zadania to dopadlo tak, že vždy sa našiel jedinec, čo pohrabal celú záhradu. Údaje z 30-tich testovaní zobrazuje, v akej generácií sa našiel jedinec, čo pohrabal celú záhradu.



4. *Zhodnotenie riešenia*

Hodnoty v evolučnom algoritme, ako napríklad koľko populácie sa vyhľadí, boli vybraté metódou pokus omyl, kedy som zadávala rôzne hodnoty a sledovala, ktoré najviac viedli k úspechu. Taktiež elitarizmus bol pridaný až neskôr, kedy som zistila, že bez neho je veľmi náročné dosiahnuť úspešné pohrabanie záhrady.