Jakub Fridrich, ID: 127147
10.10.2024

Dokumentácia k Evolučnému programovaniu

Jakub Fridrich

Zadanie: Z1b, Hľadanie pokladov

Použité IDE: VS Code

Súbor s kódom: „Z1b.py“

Vstupný súbor s inštrukciami: „start_ins.txt“

Vstupné údaje

Po spustení program načíta údaje zo súboru „start_ins.txt“. Ide o údaje v nasledovnom poradí:

1. Riadok: rozmery mapky, po ktorom sa bude EA pohybovať (X * Y)
2. Riadok: štartovná pozícia X a štartovná pozícia Y
3. Riadok: počet pokladov v mapke (num_treasures, T)
4. Až T- riadok: pozície pokladov (X Y)

Okrem tohto súboru sú vstupné údaje zadané používateľom v terminály. Ide o údaje ako **počet jedincov, počet generácii, výber selekčnej metódy (0/1/2), šanca na mutáciu, počet elitných jedincov v generácii, počet skrížených jedincov a počet zmutovaných jedincov.**

Evolučný algoritmus

Skladá sa z týchto prvkov:

Jedinec / Chromozóm: Každý jedinec obsahuje 64 náhodne vygenerovaných inštrukcii, každá inštrukciou v rozmedzí od 0 po 255 kvôli zakódovaniu na 8 bitov.

Počiatočná populácia: obsahuje n- jedincov s 64 náhodne vygenerovanými inštrukciami, n je hodnota zvolená používateľom cez terminál.

Virtual Machine / Virtuálny Stroj: táto časť kódu vytvára z inštrukcií postupnosť krokov na posun po mapke.

Elitizmus: Je vybraná časť s najväčšími fitness hodnotami z aktuálnej generácie do nasledovnej.

Selekčné metódy: Sú dostupné 3 metódy selekcie rodičov: Náhodný výber, Roletový výber a Turnajový výber.

Kríženie: Je aplikované na vytvorenie diverzity v populácii, sú pospájané jedince od polovice, v prípade turnaju ide o **multi-point** kríženie.

Mutácia: Mutácia prebieha s používateľom-vyhradenou pravdepodobnosťou. Sú dostupné 3 druhy mutácii: **Výmena 2 buniek, Reverz celého bitu** a **Náhodné dosadenie.**

Jakub Fridrich, ID: 127147
10.10.2024

Ukážka mapky z vstupného súboru

Program prečíta dáta zo súboru a následne graficky aj dátovo vytvorí reprezentáciu mapky.

```
def read_matrix_file(filename):
    global num_treasures;

    with open(filename, 'r') as file:
        matrix_dimensions = file.readline().strip().split();
        X = int(matrix_dimensions[0]);  #number x
        Y = int(matrix_dimensions[1]);  #number y

        #starting position
        start_position=file.readline().strip().split();
        start_x=int(start_position[0]);  #starting x
        start_y=int(start_position[1]);  #starting y

        #number of treasures
        num_treasures=int(file.readline().strip());

        #treasure positions
        treasures=[]
        for t in range(num_treasures):
            treasure_position = file.readline().strip().split();
            treasure_x = int(treasure_position[0]);
            treasure_y = int(treasure_position[1]);
            treasures.append((treasure_x, treasure_y));
    return X, Y, (start_x, start_y), treasures;
```

```
def create_matrix(X, Y, start_position, treasures):
    # Initialize a 2D matrix with zeros
    matrix = [[0 for m in range(X)] for m in range(Y)];

    # Mark the starting position with 1
    start_x, start_y = start_position;
    matrix[start_y][start_x]=1;

    # Mark the treasures with 2
    for treasure in treasures:
        treasure_x, treasure_y=treasure;
        matrix[treasure_y][treasure_x]=2;

    return matrix
```

```
def create_grid(X,Y,matrix):
    for y in range(Y):
        for x in range (X):
            if matrix[y][x]==0:
                canvas.create_rectangle(space+tile*x,space+tile*y,space+tile*(x+1),space+tile*(y+1),fill="#09ba00");
            elif matrix[y][x]==1:
                canvas.create_rectangle(space+tile*x,space+tile*y,space+tile*(x+1),space+tile*(y+1),fill="#09ba00");
                canvas.create_text(space+tile*x+(tile//2),space+tile*y+(tile//2),text='S',font=("Arial",15,"bold"));
            else:
                canvas.create_rectangle(space+tile*x,space+tile*y,space+tile*(x+1),space+tile*(y+1),fill="#fccf03");
```

Inicializácia počiatočnej populácie

Prvá generácia je inicializovaná generovaním N jedincov, pričom každý obsahuje 64 náhodne vygenerovaných inštrukcii. Tieto inštrukcie predstavujú 8 bitov, teda sú v rozmedzí od 0 do 255.

```
#pociatocna populacia
def ga_initial_pop(generation):
    for chromosones in range(ind_amm):
        population=[];

        for cells in range(64):
            random_value4cell=random.randrange(0,256);
            population.append(random_value4cell);

        generation.append(population);

    return generation;
```

Jakub Fridrich, ID: 127147
10.10.2024

Funkcie Virtuálneho Stroja

VM slúži na „čítanie“ inštrukcií, ktoré boli vygenerované v predošlej funkcii. Vykonáva nasledovné inštrukcie:

Inštrukcia	Tvar
Inkrementácia	00xxxxxx
Dekrementácia	01xxxxxx
Skok	10xxxxxx
Výpis	11xxxxxx

Inkrementácia znamená navýšenie hodnoty bunky, dekrementácia zníženie hodnoty bunky, skok na adresu a výpis podľa hodnoty bunky. (**vm_instructions**)

Výpis sčíta počet 1 v binárnom čísle hodnoty bunky a následne vypíše inštrukciu:

Max 2 jednotky: H

3 alebo 4 jednotky: D

5 alebo 6 jednotiek: P

7+ jednotiek: L

```
#logika vytvarania postupnosti
def vm_movemaker(bin_value):
    temp_onecounter=bin_value.count("1");
    if temp_onecounter<=2:
        ans='H';
    elif temp_onecounter==3 or temp_onecounter==4:
        ans='D';
    elif temp_onecounter==5 or temp_onecounter==6:
        ans='P';
    elif temp_onecounter==7 or temp_onecounter==8:
        ans='L';
    return ans;
```

Pohyb po mapke / hľadanie pokladov

Pohyb po mapke zabezpečuje funkcia **ga_finder**, ktorá okrem hľadania pokladov na základe postupnosti taktiež kontroluje, či nenastal pohyb mimo mapku.

Fitness funkcia

Parametre funkcie sú **počet krokov, ktoré daná postupnosť vykonala, počet nájdených pokladov a prípadná penalta.** Zvolený výpočet fitness hodnôt sa snaží odmeniť za nájdenie pokladov, pričom sa trestá za vybočenie z mapky. Premenná **found_all** je potrebná na prípadné pokračovanie v hľadaní riešení aj po nájdení všetkých pokladov.

Jakub Fridrich, ID: 127147
10.10.2024

```
def ga_fitness(steps,treasure,penalty):
    global num_treasures;
    global found_all;
    #print(treasure);
    fit=round(1-float(steps/1000)+float(treasure),4)-penalty;
    if fit==1 and steps==0:
        fit=-100;
    if treasure==num_treasures and fit>0:
        #print(treasure,num_treasures);
        #print(fit);
        found_all=True;
    return fit;
```

Výber elitných jedincov

Vykonáva sa pomocou funkcie **ga_biggest_num,** ktorá zoradí dočasnú naklonovanú generáciu a vráti elitu. Táto elita sa následne zapíše do novej generácie. Taktiež sa aj vypočítava **fittrend**, ktorý sa aj vypisuje v konzole.

```
#Elitism
zip_dic = dict(zip(num, fit_list))
best_chrom = ga_biggest_num(zip_dic, num_elite)
print(best_chrom);

fit_trend=round(sum(fit_list)/ind_amm,3);
print(f"Fit trend: {fit_trend}");

for key in best_chrom:
    best_keys_arr.append(key)
    best_array.append(temp_gen[key])

#Add elite
new_gen = best_array.copy();
new_fit_list = [fit_list[key] for key in best_keys_arr];
```

Výber náhodných jedincov v prípade, že počet elity nebol daný

```
#Old gen
old_gen = [temp_gen[i] for i in range(ind_amm) if i not in best_keys_arr]
old_fit_list = [fit_list[i] for i in range(ind_amm) if i not in best_keys_arr]

#Add non-elite
oldies_c=random.sample(old_gen,num_old);
for i in range(len(oldies_c)):
    new_gen.append(oldies_c[i]);
```

Kríženie

To, aký typ selekcie prebieha je na voľbe používateľa:

```
#Crossover
if selection_input==0:
    ga_random_selection(temp_gen,num_cross,crossover_children);
elif selection_input==1:
    ga_rulette_selection(temp_gen, fit_list, num_cross,crossover_children);
elif selection_input==2:
    ga_trnment_selection(temp_gen, fit_list, num_cross,crossover_children);
else:
    ga_trnment_selection(temp_gen, fit_list, num_cross,crossover_children);
new_gen += crossover_children;
```

Pri **náhodnom výbere** sa náhodne vyberú 2 rodičia, pričom je zakázané aby boli náhodne vybraný 2 taký istý rodičia. Ich dieťa je zložené z prvej polovici prvého rodiča a z druhej polovici druhého rodiča.

Ak bol zvolený **roletový výber,** tak ako prvé sa vypočítajú šance na zvolenie rodiča, vyberú sa rodičia a podobne ako pri náhodnom výbere je vytvorené dieťa.

Jakub Fridrich, ID: 127147
10.10.2024

```
def ga_random_selection(population, num_selections, children):
    for t in range(num_selections):
        parent1, parent2 = random.sample(population, 2);
        child = parent1[len(parent1) // 2:] + parent2[:len(parent2) // 2];
        children.append(child);

def ga_rulette_selection(population, fitness_scores, num_selections, children):
    total_fitness = sum(fitness_scores);
    selection_probs = [fitness / total_fitness for fitness in fitness_scores];

    for t in range(num_selections):
        parent1=random.choices(population, weights=selection_probs, k=1)[0];
        parent2=random.choices(population, weights=selection_probs, k=1)[0];

        child=parent1[:len(parent1)//2]+parent2[len(parent2)//2:];
        children.append(child);

    return children;
```

Pri **turnajovom výbere** sú obaja rodičia vybraný týmto typom výberu, pričom turnaj je iba medzi 2 jedincami kvôli snahe spomaliť rýchlu konvergenciu generácií. V tomto výbere je taktiež implementované **multi-point kríženie**.

```
def ga_trnment_selection(population, fitness_scores, num_selections,children):
    for t in range(num_selections):
        tournament=random.sample(range(len(population)),2);
        best_chrom=max(tournament, key=lambda i: fitness_scores[i]);
        parent1=population[best_chrom];
        #print(f"parent1: {parent1}");

        tournament=random.sample(range(len(population)),2);
        best_chrom=max(tournament, key=lambda i: fitness_scores[i]);
        parent2=population[best_chrom];

        while parent1==parent2:
            tournament=random.sample(range(len(population)),2);
            best_chrom=max(tournament, key=lambda i: fitness_scores[i]);
            parent2=population[best_chrom];
        crossover_point1 = random.randint(1, len(parent1) // 3);
        crossover_point2 = random.randint(len(parent1) // 3, len(parent1) - 1);
        #child = parent1[:len(parent1) // 2] + parent2[len(parent2) // 2:];

        child = parent1[:crossover_point1] + parent2[crossover_point1:crossover_point2] + parent1[crossover_point2:]
        children.append(child);
```

Mutácia

To, aký jedinci zmutujú sa vyberá zo **skrížených detí** nasledovne:

```
    #Mutation
    mut_prep=[];
    mut_prep = random.sample(range(len(crossover_children)), num_mut)#indexy jedincov

    for child in mut_prep:
        mutated_child = mut_switch_add(crossover_children[child], mut_rate);
        mutated_child = mut_rand_value(mutated_child, mut_rate);
        mutated_child = mut_invert_bits(mutated_child, mut_rate);
        mut_indiv.append(mutated_child);
    #print(f"Pridaných {len(mut_indiv)} mutovaných jedincov!")
    new_gen += mut_indiv
```

Sú dostupné 3 druhy mutácií: **dosadenie náhodnej hodnoty, výmena 2 buniek, reverz bitov.**

Jakub Fridrich, ID: 127147
10.10.2024

```
def mut_rand_value(population, rate):
    temp_array=copy.deepcopy(population)

    for ind in range(len(population)):
        if random.random()<rate:
            rand_value_replace=random.randrange(0, 255);
            temp_array[ind]=rand_value_replace;
    return temp_array;

def mut_switch_add(population, rate):
    temp_array=copy.deepcopy(population);

    for ind in range(len(population)):
        if random.random()<rate:
            rand_ad_switch=random.randrange(0,64);
            while ind == rand_ad_switch:
                rand_ad_switch=random.randrange(0,64);
            temp_array[ind],temp_array[rand_ad_switch]=temp_array[rand_ad_switch],temp_array[ind];
    return temp_array;

def mut_invert_bits(population, rate):
    temp_array=copy.deepcopy(population);

    for ind in range(len(population)):
        if random.random()<rate:
            bin_chrom="{:0>8}".format(bin(temp_array[ind])[2:]);
            rev_bin=bin_chrom[::-1];
            temp_array[ind]=int(rev_bin,2);
            #print(population[ind], bin_chrom, rev_bin, int(rev_bin,2),temp_array[ind]);
    return temp_array;
```

Výpis najlepšieho riešenia

Aby sa zaistilo vypísanie skutočne najlepšieho riešenia, program neustále kontroluje či sa medzi elitou nachádza lepší fitness.

```
        if fitness > best_fitness:
            best_fitness = fitness;
            best_sequence = postupnost.copy();
```

Tento najlepší fitness aj s jeho postupnosťou je nakoniec vypísaný na konci programu:

```
print(f"Best Fitness: {best_fitness}");
print(f"Best Sequence of Moves: {best_sequence}");
```

Podmienky hľadania

Program sa zastaví, akonáhle bude splnená niektorá z nasledovných podmienok:

1. program našiel všetky poklady

```
def ga_fitness(steps,treasure,penalty):
    global num_treasures;
    global found_all;
    #print(treasure);
    fit=round(1-float(steps/1000)+float(treasure),4)-penalty;
    if fit==1 and steps==0:
        fit=-100;
    if treasure==num_treasures and fit>0:
        #print(treasure,num_treasures);
        #print(fit);
        found_all=True;
    return fit;
```

```
        if found_all==True and con_after==False:
            con_choice=int(input(f"Vsetky poklady najdene - {treasure} - chcete pokracovat? (0/1):"));
            if con_choice==0:
                break;
            else:
                con_after=True;
```

2. postupnosť, generovaná programom, vybočila zo stanovenej mriežky

Jakub Fridrich, ID: 127147
10.10.2024

```
while i != len(sequence):
    #print(sequence[i], position_x,position_y)
    if sequence[i]=='P':
        if position_x+1>6:
            penalty+=100;
            break;
        else:
            position_x+=1;
    elif sequence[i]=='L':
        if position_x-1<0:
            penalty+=100;
            break;
        else:
            position_x-=1;
    elif sequence[i]=='D':
        if position_y+1>6:
            penalty+=100;
            break;
        else:
            position_y+=1;
    elif sequence[i]=='H':
        if position_y-1<0:
            penalty+=100;
            break;
        else:
            position_y-=1;
    steps+=1;
```

3. program vykonal 500 krokov (inštrukcií)

```
for pop in range(ind_amm):
    for ins_limit in range(1000):
        if pointer_ad>63 or len(postupnost)>500:
            break
        clone_gen[pop] = vm_instructions(clone_gen[pop]);
```

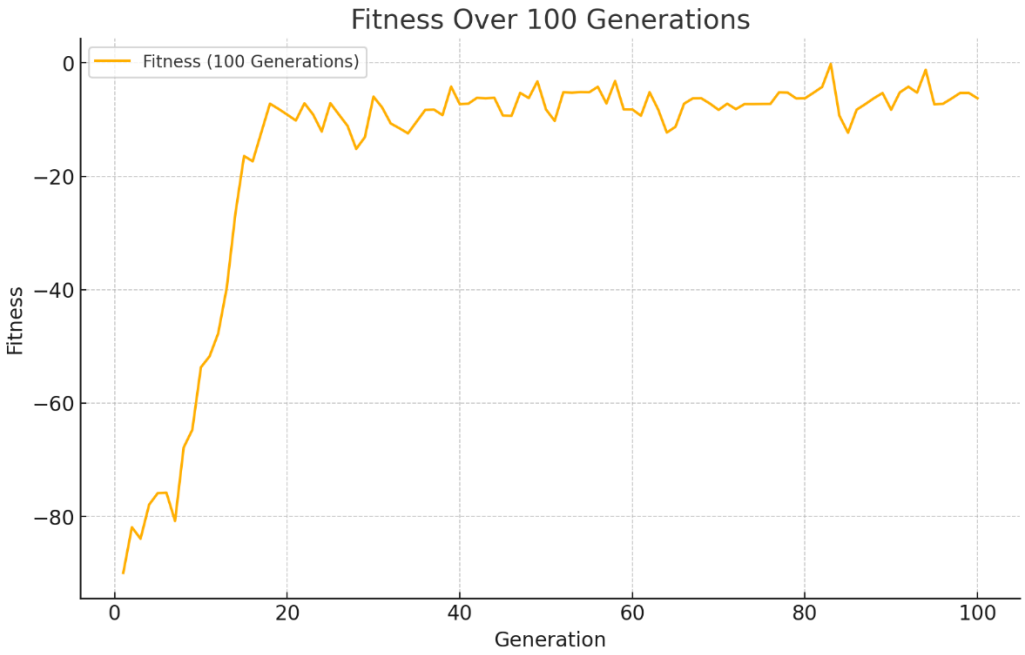
Jakub Fridrich, ID: 127147
10.10.2024

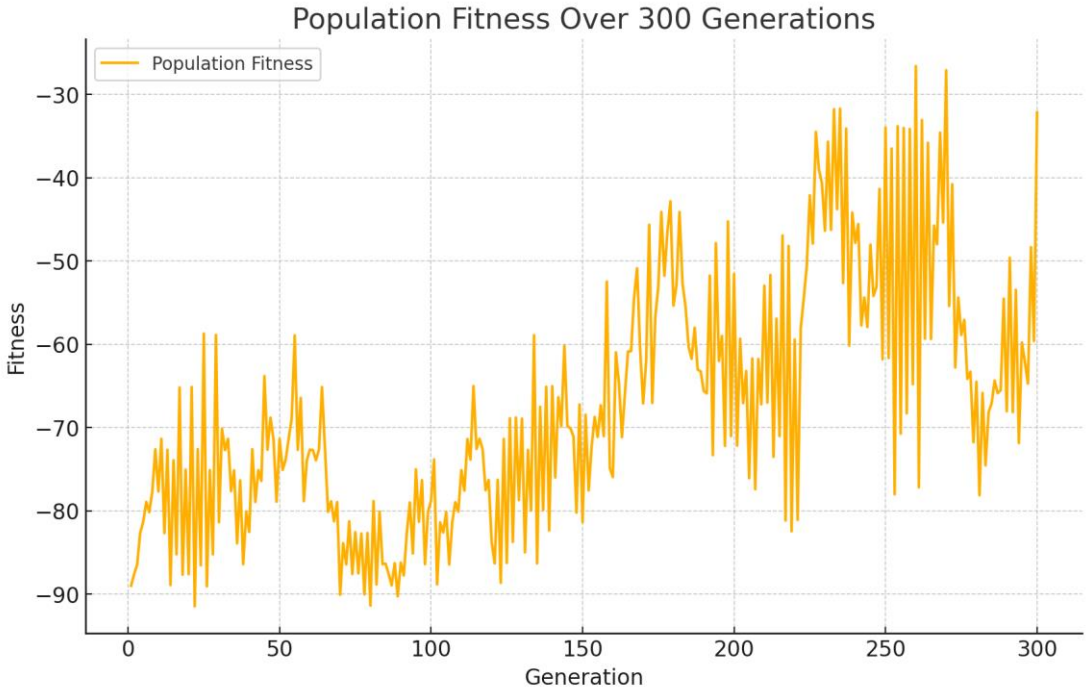
Porovnanie výsledkov

Pre demonštráciu kódu som sa rozhodol porovnať výsledky dvoch rôznych druhov selekcií- turnaj a náhodný výber. V prvom prípade bola veľkosť populácie 100 a počet generácií 100, v druhom bol počet generácií 300 s rovnako veľkou šancou na mutáciu a mutačnými spôsobmi.

Porovnaním týchto dvoch riešení je možné vidieť, že pokus s turnajom rýchlejšie konvergoval a napokon stagnoval. Pri pokuse s náhodným výberom je vidieť, že takýto výber je oveľa neefektívnejší, nakoľko našlo sa málo pokladov a fit trend nestúpol ponad zápornú hranicu.

Pri oboch pokusoch boli nájdené 3 poklady z 5.





Jakub Fridrich, ID: 127147
10.10.2024

Dolaďovanie riešenia

Pri vypracovaní zadania som dospel k pár poznatkom:

Väčšia populácia značne vylepšuje graf fit trendu pri používaní turnajového výberu, avšak najlepšiu rovnováhu medzi kvalitou grafu, riešenia a času zahŕňa používať mutačný rate 5%, Elitizmus o veľkosti 5%, krížencov o veľkosti 80% a mutantov o veľkosti 15%.

Miesta kde vidím priestor na zlepšenie sú zmena mutačného rate-u, poprípade zmeny v mutáciách, ako sú typy mutácií a ich poradie.