Slovenská technická univerzita

Fakulta informatiky a informačných technológií

Ilkovičova 3, 842 19 Bratislava 4

Sakalosh Oleksandr

Problém obchodného cestujúceho

Študijný program: Informatika

Ročník: 3.

Predmet: **Umelá inteligencia**

Ak. rok: 2021/2022

ÚLOHA

Obchodný cestujúci má navštíviť viacero miest. V jeho záujme je minimalizovať cestovné náklady a cena prepravy je úmerná dĺžke cesty, preto snaží sa nájsť najkratšiu možnú cestu tak, aby každé mesto navštívil práve raz. Keďže sa nakoniec musí vrátiť do mesta z ktorého vychádza, jeho cesta je uzavretá krivka.

ZADANIE

Je daných aspoň 20 miest (20 – 40) a každé má určené súradnice ako celé čísla X a Y. Tieto súradnice sú náhodne vygenerované. (Rozmer mapy môže byť napríklad 200 * 200 km.) Cena cesty medzi dvoma mestami zodpovedá Euklidovej vzdialenosti – vypočíta sa pomocou Pytagorovej vety. Celková dĺžka trasy je daná nejakou permutáciou (poradím) miest. Cieľom je nájsť takú permutáciu (poradie), ktorá bude mať celkovú vzdialenosť čo najmenšiu.

Výstupom je poradie miest a dĺžka zodpovedajúcej cesty.

GENETICKÝ ALGORITMUS

Genetická informácia je reprezentovaná vektorom, ktorý obsahuje index každého mesta v nejakom poradí (nejaká permutácia miest). Keďže hľadáme najkratšiu cestu, je najlepšie vyjadriť fitnes jedinca ako prevrátenú hodnotu dĺžky celej cesty.

Jedincov v prvej generácii inicializujeme náhodne – vyberáme im náhodnú permutáciu miest. Jedincov v generácii by malo byť tiež aspoň 20. Je potrebné implementovať aspoň dve metódy výberu rodičov z populácie.

Kríženie je možné robiť viacerými spôsobmi, ale je potrebné zabezpečiť, aby vektor génov potomka bol znovu permutáciou všetkých miest. Často používaný spôsob je podobný dvojbodovému kríženiu. Z prvého rodiča vyberieme úsek cesty medzi dvoma náhodne zvolenými bodmi kríženia a dáme ho do potomka na rovnaké miesto. Z druhého rodiča potom vyberieme zvyšné mestá v tom poradí, ako sa nachádzajú v druhom rodičovi a zaplníme tým ostatné miesta vo vektore.

Mutácie potomka môžu byť jednoduché – výmena dvoch susedných miest alebo zriedkavejšie používaná výmena dvoch náhodných miest. Tá druhá výmena sa používa zriedkavo, lebo môže rozhodiť blízko optimálne riešenie. Často sa však používa obrátenie úseku – znova sa zvolia dva body a cesta medzi nimi sa obráti. Sú možné aj ďalšie mutácie, ako napríklad posun úseku cesty niekam inam.

Gen je objekt triedy **Gen** a uchováva informácie o poradí miest, fitness (prejdena vzdialenosť) a možnosti jeho prechodu na ďalšiu populáciu.

```
@dataclass
class Gen:
    order: list
    fitness: float
    prob: float
```

Prvá populácia sa vytvorí takto:

Vytvoríme prázdne pole populácie, cez cyklus for k nemu pridáme požadovaný počet génov. Na vytvorenie génu vyplňme prázdne pole číslami od 0 do počtu miest, náhodne ich zmiešajme a vypočítajme dĺžku cesty.

```
population = []
for i in range(POP_SIZE):
    gen = generateFirst()
    population.append(gen)
population = calcProb(population)
```

```
def generateFirst():
    global COUNT
    order = []
    for i in range(1, COUNT):
        order.append(i)

    order = random.sample(order,len(order))
    fitn = calcFitn(order)
    gen = Gen(order, fitn, 0)
    return gen
```

Novú generáciu vytváram skopírovaním tej starej a zmenou génov v nej. Prvých 25% v novej generácii sú mutované náhodné gény z predchádzajúcej. Ďalších 25 % sú zmutované kópie najlepšieho génu z predchádzajúcej. Ďalších 50 % tvoria skrížené gény tých najlepších a náhodných z predchádzajúcej.

```
def createNew(population):
    global mutationRate
    if mutationRate > 0.1:
        mutationRate = mutationRate1/1.1
    old = copy.deepcopy(population)
    population.clear()
    for i in range(POP_SIZE//4):
        gen1 = copy.deepcopy(pickOneRandom(old))
        population.append(mutate(gen1, mutationRate))

    for i in range(POP_SIZE//4, POP_SIZE//2):
        gen2 = copy.deepcopy(pickBest(old))
        population.append(mutate(gen2, mutationRate))

    for i in range(POP_SIZE//2, POP_SIZE):
        gen3 = copy.deepcopy(pickBest(old))
        gen4 = copy.deepcopy(pickOneRandom(old))
        gen = crossOver(gen3, gen4)
        population.append(mutate(gen, mutationRate))

    population = calcProb(population)

    return population
```

Funkcia **pickOneRandom()** vyberie gén, pričom venuje pozornosť tomu, ktorý z nich má väčšiu šancu.

```
def pickOneRandom(population):
    pop = population.copy()
    r = random.random()
    i = 0
    while r>0:
        r -= pop[i].prob
        i+=1

    i-=1
    gen1 = pop[i]

    return gen1
```

Funkcia **mutate()** vyberie náhodné mesto v géne a zmení jeho poradie s ďalším susedom, ak náhodné číslo nie je vyššie ako mutRate.

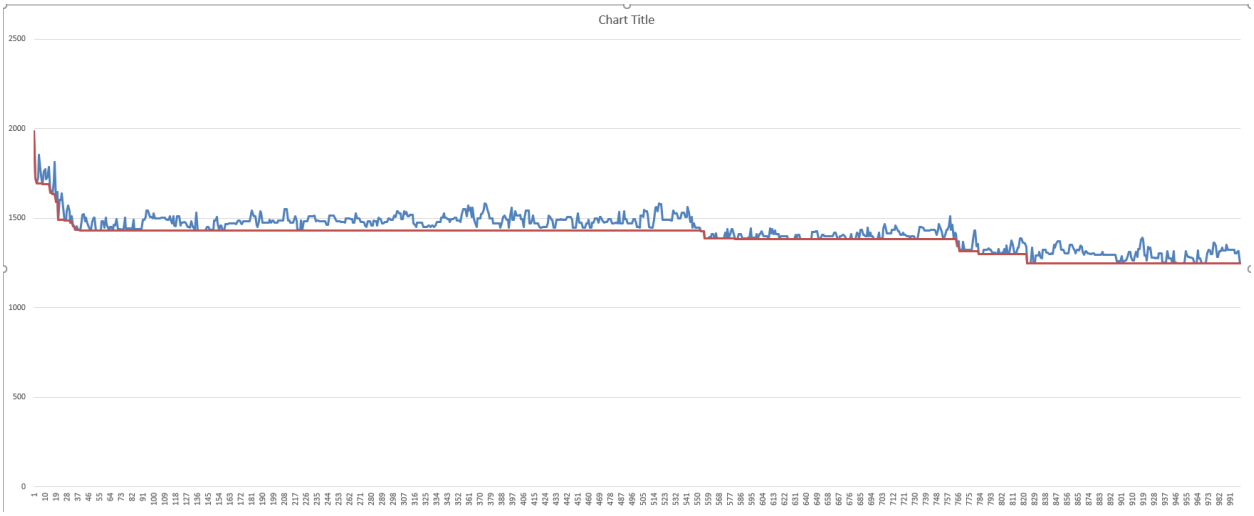
```
def mutate(gen, mutRate):
    global BEST_RES
    k = 0
    for i in range(COUNT):
        g = random.random()
        if g < mutRate:
            k = 1
            r1 = random.choice(gen.order)
            i1 = gen.order.index(r1)
            i2 = (i1 + 1) % (COUNT - 1)
            r2 = gen.order[i2]
            gen.order[i1] = r2
            gen.order[i2] = r1

    gen.fitness = calcFitn(gen.order)
    if gen.fitness < BEST_RES.fitness:
        BEST_RES = gen
    return gen
```

Funkcia **crossOver()** vyberie 2 náhodné mestá v prvom géne, skopíruje do nového génu všetky mestá medzi nimi a pridá všetky ostatné mestá v poradí, v akom idú v druhom géne.

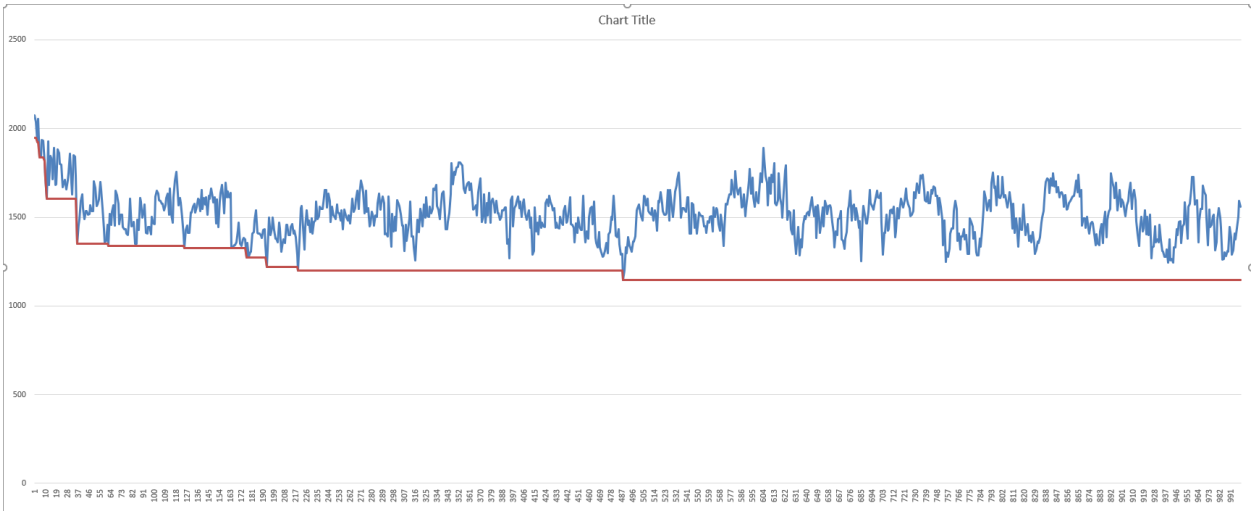
```
def crossOver(gen1, gen2):
    r = random.sample(gen1.order, 2)
    i1 = gen.order.index(r[0])
    i2 = gen.order.index(r[1])
    kid = Gen(gen1.order[i1:i2], 0, 0)
    for city in gen2.order:
        if city not in kid.order:
            kid.order.append(city)
    return kid
```

Porovnanie dosahovaných výsledkov pre dva rôzne spôsoby tvorby novej generácie

1. Iba mutácia



2. Iba skríženie



3. Kombinované