demo-02_population

December 14, 2020

1 Analiza i projektiranje računalom - 4. laboratorijska vježba: demo population.py

1.1 Priprema za izvođenje

[1]:
```
import os

CD_KEY = "--HW04_D02_IN_ROOT"
```

[2]:
```
if (
    CD_KEY not in os.environ
    or os.environ[CD_KEY] is None
    or len(os.environ[CD_KEY]) == 0
    or os.environ[CD_KEY] == "false"
):
    %cd ..
else:
    print(os.getcwd())

os.environ[CD_KEY] = "true"
```

```
/mnt/data/projekti/faks/AIPR/dz/dz-04
```

1.2 Učitavanje paketa

[3]:
```
import numpy as np

from src.evolution.function import Function
from src.evolution.population import Population
```

1.3 Inicijalizacija

1.3.1 Formatiranje

[4]:
```
np.set_printoptions(precision=2, suppress=True)
```

1.3.2 Konstante

[5]:
```
wellness_function = Function(lambda x: np.mean(np.square(x)))
capacities = (5, 10)
```

1.3.3 Populacije

[6]:
```
populations = [
    Population(
        wellness_function=wellness_function,
        capacity=capacity
    )
    for capacity in capacities
]
```

1.4 Demonstracija

[7]:
```
for population in populations:
    print(population, end="\n\n")
```

```
Population (0 / 5)

Population (0 / 10)
```

[8]:
```
for population in populations:
    for _ in range(capacities[0]):
        population.add(np.random.uniform(-1, 1, (3,)))

    print(population, end="\n\n")
```

```
Population (5 / 5)
        [0.405467043743754] [-0.61 -0.88  0.27]
        [0.3926336403309582] [-0.94  0.48 -0.25]
        [0.3706202121692716] [-0.64  0.72 -0.42]
        [0.3660067466501335] [ 0.75 -0.49 -0.54]
        [0.32095516628613235] [-0.16  0.96  0.12]

Population (5 / 10)
        [0.4176191801577615] [-0.45  0.96  0.34]
        [0.4156262991454048] [ 0.87 -0.61  0.33]
        [0.3318631981345519] [ 0.71  0.05 -0.7 ]
        [0.07304104727942949] [-0.03  0.46 -0.09]
        [0.02753187426624325] [ 0.01  0.28 -0.06]
```

Ako pokušamo dodati primjerak s `add`, a popunjeni su kapaciteti populacije, prvo će se maknuti najgori primjerak, a onda umetnuti novi.

[9]:
```
for population in populations:
    population.add(np.array([1, 1, 1]))

    print(population, end="\n\n")
```

```
Population (5 / 5)
        [1.0] [1 1 1]
        [0.405467043743754] [-0.61 -0.88  0.27]
        [0.3926336403309582] [-0.94  0.48 -0.25]
        [0.3706202121692716] [-0.64  0.72 -0.42]
        [0.3660067466501335] [ 0.75 -0.49 -0.54]

Population (6 / 10)
        [1.0] [1 1 1]
        [0.4176191801577615] [-0.45  0.96  0.34]
        [0.4156262991454048] [ 0.87 -0.61  0.33]
        [0.3318631981345519] [ 0.71  0.05 -0.7 ]
        [0.07304104727942949] [-0.03  0.46 -0.09]
        [0.02753187426624325] [ 0.01  0.28 -0.06]
```

Komentar: Podrazumijevani argument `remove_before` je postavljen na `True`, čime se osigurava da prilikom prepunjene populacije mičemo najgori element. Međutim, to ne garantira da će u populaciji ostati i **najbolji** element. Ako želimo garantirati i to, moramo postaviti `remove_before` na `False` - ovime se omogućava privremena prepopulacija, a tek se nakon svih unosa brišu dodatni elementi počevši od najgoreg. Kasnije ćemo demonstrirati ovo ponašanje.

Ako želimo dodavati elemente u populaciju bez da automatski mičemo višak, to možemo raditi koristeći `append`.

[10]:
```
for population in populations:
    population.append(np.array([-1, -1, -1]))

    print(population, end="\n\n")
```

```
Population (6 / 5)
        [1.0] [1 1 1]
        [1.0] [-1 -1 -1]
        [0.405467043743754] [-0.61 -0.88  0.27]
        [0.3926336403309582] [-0.94  0.48 -0.25]
        [0.3706202121692716] [-0.64  0.72 -0.42]
        [0.3660067466501335] [ 0.75 -0.49 -0.54]

Population (7 / 10)
        [1.0] [1 1 1]
        [1.0] [-1 -1 -1]
        [0.4176191801577615] [-0.45  0.96  0.34]
        [0.4156262991454048] [ 0.87 -0.61  0.33]
        [0.3318631981345519] [ 0.71  0.05 -0.7 ]
```

```
        [0.07304104727942949] [-0.03  0.46 -0.09]
        [0.02753187426624325] [ 0.01  0.28 -0.06]
```

Višak elemenata možemo obrisati pozivom `cull` nad populacijom.

[11]:
```
for population in populations:
    population.cull()

    print(population, end="\n\n")
```

```
Population (5 / 5)
        [1.0] [1 1 1]
        [1.0] [-1 -1 -1]
        [0.405467043743754] [-0.61 -0.88  0.27]
        [0.3926336403309582] [-0.94  0.48 -0.25]
        [0.3706202121692716] [-0.64  0.72 -0.42]

Population (7 / 10)
        [1.0] [1 1 1]
        [1.0] [-1 -1 -1]
        [0.4176191801577615] [-0.45  0.96  0.34]
        [0.4156262991454048] [ 0.87 -0.61  0.33]
        [0.3318631981345519] [ 0.71  0.05 -0.7 ]
        [0.07304104727942949] [-0.03  0.46 -0.09]
        [0.02753187426624325] [ 0.01  0.28 -0.06]
```

Metoda `cull` ima pretpostavljen argument `n_additional` postavljen na `0`. Ako želimo obrisati više ili manje elemenata od broja definiranim kapacitetom, to možemo promijenom argumenta. Na primjer, ako želimo da ostanu 3 elementa manje od kapaciteta, onda možemo pisati

[12]:
```
for population in populations:
    population.cull(3)

    print(population, end="\n\n")
```

```
Population (2 / 5)
        [1.0] [1 1 1]
        [1.0] [-1 -1 -1]

Population (7 / 10)
        [1.0] [1 1 1]
        [1.0] [-1 -1 -1]
        [0.4176191801577615] [-0.45  0.96  0.34]
        [0.4156262991454048] [ 0.87 -0.61  0.33]
        [0.3318631981345519] [ 0.71  0.05 -0.7 ]
        [0.07304104727942949] [-0.03  0.46 -0.09]
        [0.02753187426624325] [ 0.01  0.28 -0.06]
```

Možemo izbaciti specifične primjerke koristeći metodu `ban`, ako postoje u populaciji.

[13]:
```
for population in populations:
    population.ban(np.array([1, 1, 1]))

    print(population, end="\n\n")
```

```
Population (1 / 5)
        [1.0] [-1 -1 -1]

Population (6 / 10)
        [1.0] [-1 -1 -1]
        [0.4176191801577615] [-0.45  0.96  0.34]
        [0.4156262991454048] [ 0.87 -0.61  0.33]
        [0.3318631981345519] [ 0.71  0.05 -0.7 ]
        [0.07304104727942949] [-0.03  0.46 -0.09]
        [0.02753187426624325] [ 0.01  0.28 -0.06]
```

Radi demonstracije dodat ćemo jedan element natrag.

[14]:
```
populations[0].add(np.array([1, 1, 1]))
print(populations[0])
```

```
Population (2 / 5)
        [1.0] [-1 -1 -1]
        [1.0] [1 1 1]
```

Također, možemo i maknuti specifični indeks iz populacije uz `pop`. Podrazumijevani indeks je `-1`, tj. najgora jedinka.

[15]:
```
for population in populations:
    population.pop(1)

    print(population, end="\n\n")
```

```
Population (1 / 5)
        [1.0] [-1 -1 -1]

Population (5 / 10)
        [1.0] [-1 -1 -1]
        [0.4156262991454048] [ 0.87 -0.61  0.33]
        [0.3318631981345519] [ 0.71  0.05 -0.7 ]
        [0.07304104727942949] [-0.03  0.46 -0.09]
        [0.02753187426624325] [ 0.01  0.28 -0.06]
```

Slično kao i s `append`, možemo dodati kolekciju jedinki koristeći `assimilate`.

[16]:
```
for population in populations:
    population.assimilate(np.random.uniform(-1, 1, (10, 3)))

    print(population, end="\n\n")
```

```
Population (11 / 5)
        [1.0] [-1 -1 -1]
        [0.6042271036232179] [-0.91  0.28 -0.95]
        [0.43310414228301014] [ 0.99  0.38 -0.4 ]
        [0.36840489358039297] [ 0.93 -0.28 -0.41]
        [0.30868505607536634] [ 0.23 -0.48  0.8 ]
        [0.2647809916340921] [ 0.15 -0.72 -0.5 ]
        [0.26372528814757135] [-0.63  0.59 -0.24]
        [0.22964932621797618] [ 0.48 -0.29 -0.61]
        [0.21296621385043077] [-0.24  0.72 -0.25]
        [0.17394078848858463] [0.51 0.28 0.43]
        [0.05148656636232918] [-0.12  0.22 -0.3 ]

Population (15 / 10)
        [1.0] [-1 -1 -1]
        [0.6967082858420514] [-0.92  0.87 -0.69]
        [0.5167601368076115] [-0.8  -0.88  0.39]
        [0.5007047583971788] [ 0.45 -0.77 -0.84]
        [0.4156262991454048] [ 0.87 -0.61  0.33]
        [0.3503467574542702] [0.41 0.24 0.91]
        [0.3318631981345519] [ 0.71  0.05 -0.7 ]
        [0.19521496766992685] [0.46 0.26 0.55]
        [0.1651114266390732] [-0.69 -0.12  0.01]
        [0.07304104727942949] [-0.03  0.46 -0.09]
        [0.07119841851388475] [-0.07 -0.38  0.25]
        [0.0696216669988322] [-0.31  0.25 -0.22]
        [0.05016548789413151] [-0.32  0.2   0.07]
        [0.02753187426624325] [ 0.01  0.28 -0.06]
        [0.0017372741122927537] [-0.01  0.07  0.  ]
```

Za funkcionalnost analognu `add`, možemo dodati kolekciju uz provjeru kapaciteta s `invade`. Prvo ćemo to napraviti tako da postavimo `remove_before` na `False`.

[17]:
```
for population in populations:
    population.invade(np.random.uniform(-1, 1, (10, 3)), remove_before=False)

    print(population, end="\n\n")
```

```
Population (5 / 5)
        [1.0] [-1 -1 -1]
        [0.6355914802131851] [0.97 0.97 0.2 ]
        [0.6042271036232179] [-0.91  0.28 -0.95]
```

```
        [0.5523923483513739] [-0.57 -0.8   0.84]
        [0.5352335545086736] [-0.74  0.89  0.53]

Population (10 / 10)
        [1.0] [-1 -1 -1]
        [0.6967082858420514] [-0.92  0.87 -0.69]
        [0.5167601368076115] [-0.8  -0.88  0.39]
        [0.5007047583971788] [ 0.45 -0.77 -0.84]
        [0.4972287914624967] [-0.82  0.9   0.01]
        [0.4156262991454048] [ 0.87 -0.61  0.33]
        [0.39063215445572613] [0.71 0.82 0.01]
        [0.3782676864591415] [-0.99  0.39 -0.01]
        [0.3503467574542702] [0.41 0.24 0.91]
        [0.3318631981345519] [ 0.71  0.05 -0.7 ]
```

Komentar: Vidimo da su najbolje jedinke očuvane ako su ostale najbolje.

Ako ovu operaciju primijenimo bez gorenavedene promjene, onda čuvanje najboljeg elementa nije garantirano.

[18]:
```
for population in populations:
    population.invade(np.random.uniform(-1, 1, (10, 3)))

    print(population, end="\n\n")
```

```
Population (5 / 5)
        [0.6085313911729354] [-0.72 -0.86 -0.75]
        [0.5375830735701992] [0.78 0.92 0.41]
        [0.4189748491810332] [-0.45  0.86  0.56]
        [0.3186627576894004] [ 0.51 -0.76  0.34]
        [0.2830583089403191] [ 0.24  0.39 -0.8 ]

Population (10 / 10)
        [0.8325515420299371] [ 0.93  0.96 -0.85]
        [0.8175685848789844] [ 0.88 -0.86  0.97]
        [0.537456559775566] [0.87 0.75 0.54]
        [0.5119651542453343] [ 0.44 -0.65 -0.96]
        [0.49725566515443886] [0.92 0.59 0.55]
        [0.4244774661231203] [0.67 0.84 0.34]
        [0.3720800947487921] [-0.07 -0.35  0.99]
        [0.3559374168926856] [-0.26 -0.31 -0.95]
        [0.2387080933140102] [-0.19 -0.24 -0.79]
        [0.0915815012799288] [ 0.09 -0.46  0.24]
```

1.4.1 Mehanike elitizma

Moguće je implementirati i elitizam, no tada populacija ima nešto drukčije ponašanje, koje može biti neželjeno. Podrazumijevana vrijednost elitizma u populaciji je `0`, a ona ne bi trebala biti negativan broj (u slučaju da je željeno ponašanje micanje najboljih jedinki to se treba implementirati na neki drugi, jasniji način).

[19]:
```
elite_capacity = 5
elite_elitism = 1
```

[20]:
```
elite_population = Population(
    wellness_function=wellness_function,
    capacity=elite_capacity,
    elitism=elite_elitism
)
elite_population.assimilate(np.random.uniform(-1, 1, (5, 3)))
```

[21]:
```
print(elite_population)
```

```
Population (5 / 5), elite: 1
        [0.5080701673617466] [ 0.86 -0.88  0.07]
        [0.34897078584204216] [ 0.86  0.52 -0.18]
        [0.27835349623182387] [0.26 0.64 0.6 ]
        [0.13509128390281058] [ 0.33 -0.38  0.39]
        [0.11060165125162276] [ 0.51  0.12 -0.24]
```

Kad bi htjeli maknuti sve elemente, najboljih `elitism` će ostati.

[22]:
```
elite_population.cull(elite_population.capacity)
print(elite_population)
```

```
Population (1 / 5), elite: 1
        [0.5080701673617466] [ 0.86 -0.88  0.07]
```

Ako pokušamo iskoristiti `pop` u ovom slučaju, vratit će element koji tražimo, ali ga neće maknuti iz populacije.

[23]:
```
print(f"Traženi element: {elite_population.pop()}\n")
print(elite_population)
```

```
Traženi element: [0.5080701673617466] [ 0.86 -0.88  0.07]

Population (1 / 5), elite: 1
        [0.5080701673617466] [ 0.86 -0.88  0.07]
```

Međutim, postoji posebni slučaj koji nije reguliran elitizmom, a to je `ban`. U slučaju pozivanju `ban` nad elitom, moguće je čak prekršiti i ograničenje minimalnog broja jedinki.

[24]:
```
elite_population.ban(elite_population[0])
print(elite_population)
```

```
Population (0 / 5), elite: 1
```

Komentar: Ovakvo ponašanje je namjerno kako bi postojao način pražnjenja populacije s elitizmom.