Biološki inspirisani algoritmi – Prvi domaći zadatak

Koristeći ***Genetski algoritam*** rešićemo poznati problem ***Trgovačkog putnika***, u *Python-u* uz pomoć *deap* biblioteke. Problem Trgovačkog putnika glasi:
Zadata je lista Gradova kao I udaljenost između svih gradova. Treba pronaći najkraći mogući put kojim se obilaze svi gradovi, nakon čega se putnik mora vratiti u grad iz koga je krenuo. Pritom svaki grad sme biti posećen samo jednom. Najpre definišemo funkciju koja će učitati vrenosti za zadate gradove sa sledećeg web sajta: http://elib.zib.de/pub/mp-testdata/tsp/tsplib/tsp/. Koristićemo 29 Bavarskih gradova I njihove koordinate kao ulaz.

Moramo definisati parametre našeg algoritma:

- NUM_GENERATIONS: broj generacija koje ćemo iterisati u potrazi za optimalnim rešenjima.
- POPULATION_SIZE: veličina populacije u smislu potencijalnih rešenja koju ćemo koristiti u svakoj generaciji.
- P_CROSSOVER: verocatnoća selektovanja jedinke za parenje (crossover)
- P_MUTATUION: verovatnoća selektovanja jedinke za nasumičnu mutaciju

Zatim nasumično biramo jednu putanju kao potencijalno rešenje:

```
individual = random.sample(individual, len(individual))
```

koju možemo prikazati preko individualnih indeksa svakkog grada iz array liste:

[16, 17, 20, 8, 23, 24, 10, 7, 26, 1, 5, 4, 11, 25, 21, 14, 3, 0, 19, 22, 12, 18, 2, 15, 13, 28, 9, 6, 27]

Definišemo funkciju koja će računati udaljenost za određenu izabranu putanju, kao i *FitnessMin* tip sa negativnom težinom iz razloga minimizacije udaljenosti. Kreiramo operator koji će nasumično promešati naše gradove, zarad kreiranja nove nasumične putanje (jedinke), kako bismo ispunili zadatu populaciju nasumičnim jedinkama.

Nakon definisanja funkcije koja računa Fitness (dobtoru) jedinke, postavljamo esencijalne operatore za naš Genetički algoritam:

- evaluate: računa Fitness jedinke
```
toolbox.register('evaluate', tspFitness)
```
- select: bira jedinke koje će se pariti da bi se dobilo novo potomstvo
```
toolbox.register('select', tools.selTournament, tournsize=3)
```
- mate: vrši ukrštanje na prethodno odabranim jedinkama.
```
toolbox.register('mate', tools.cxOrdered)
```
- mutate: bira jedinku za mutaciju. Verovatnoća mutacije se izračunava tako da se najmanje jedan indeks izmeša.
```
toolbox.register('mutate', tools.mutShuffleIndexes, indpb=1.0 / NUMBER_CITIES)
```

HallOfFame objekat koristimo za čuvanje najboljih pojedinaca svake generacije, koristićemo ga za sprovođenje elitizma tako što ćemo uzeti najbolje pojedince iz svake generacije i promovisati ih u sledeću generaciju.

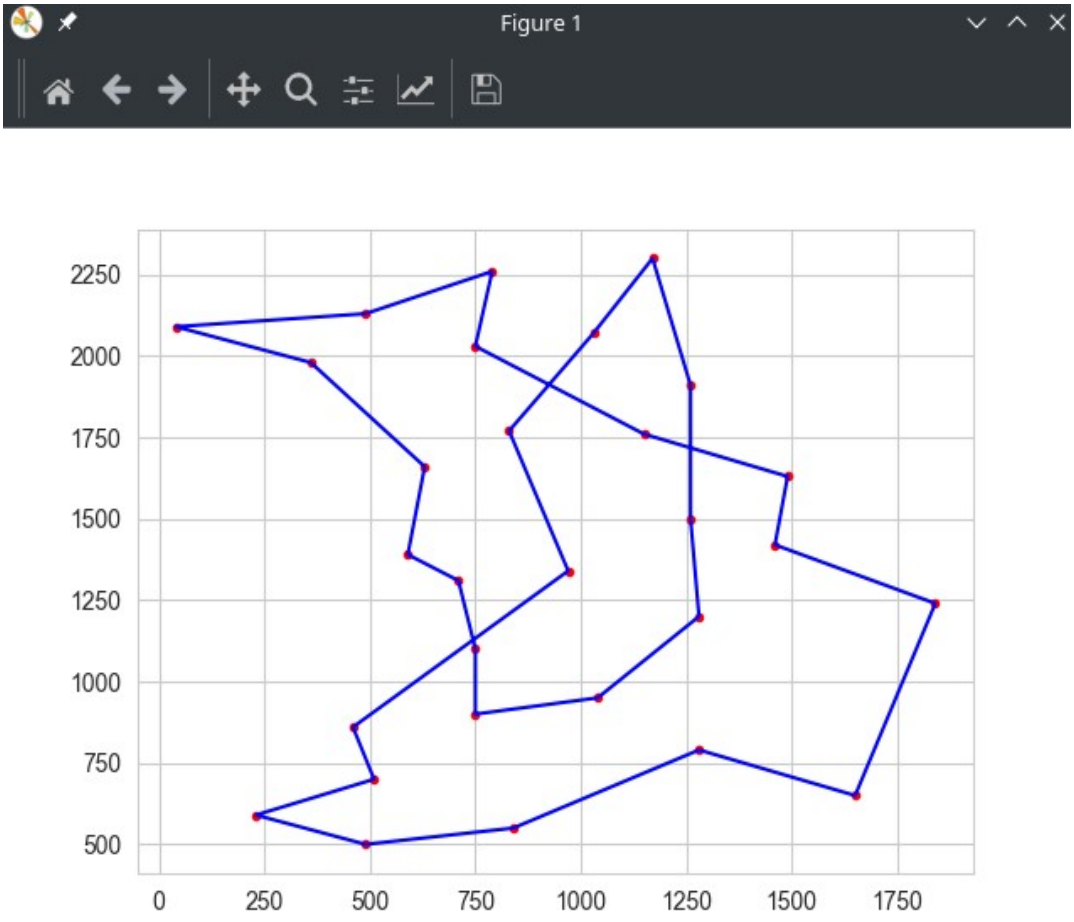
Koristimo *Statistics* objekat za praćenje nekoliko statistika pojedinaca.

```
stats = tools.Statistics(lambda ind: ind.fitness.values)
stats.register('min', np.min)
stats.register('avg', np.mean)
```

Logbook objekat se koristi za praćenje statistike populacije svake generacije.

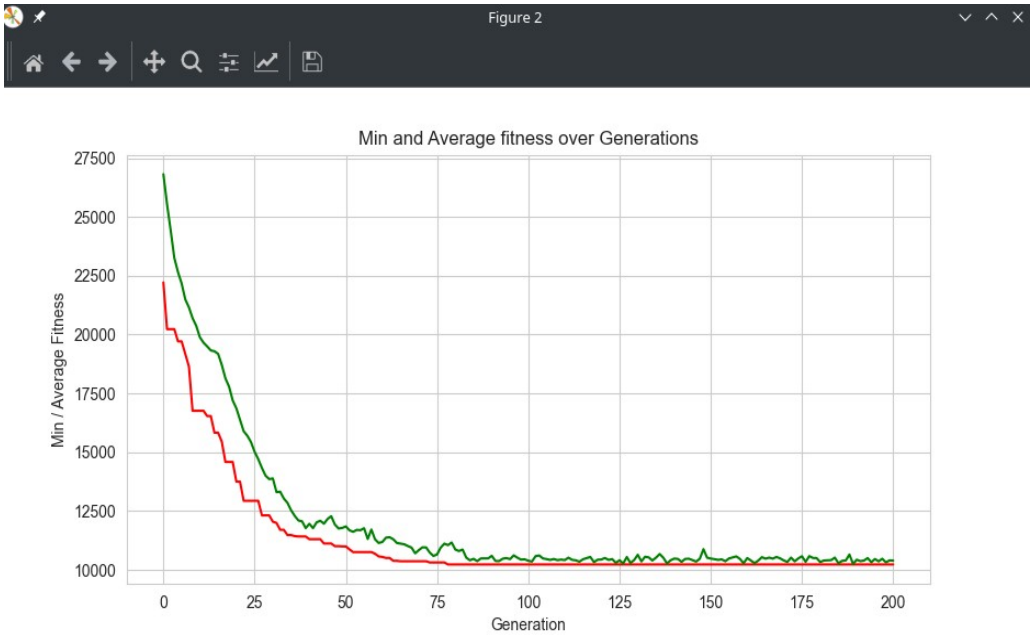
```
logbook = tools.Logbook()
logbook.header = ['gen', 'nevals'] + stats.fields
```

Sada smo spremni da započnemo iterativni genetski tok, kak bismo evoluirali jedinke iz populacije i pronašli optimalno rešenje. Nakon izvrešnog algoritma možemo da izvučemo najboljeg pojedinca iz *HallOfFame* objekta i iscrtamo optimalan put.
Ovako izgleda najbolja putanja koju je Genetički algoritam uspeo da pronađe tokom 200 generacija:



Putanja 1: POPULATION_SIZE=100, P_CROSSOVER=0.9, P_MUTATUION=0.1

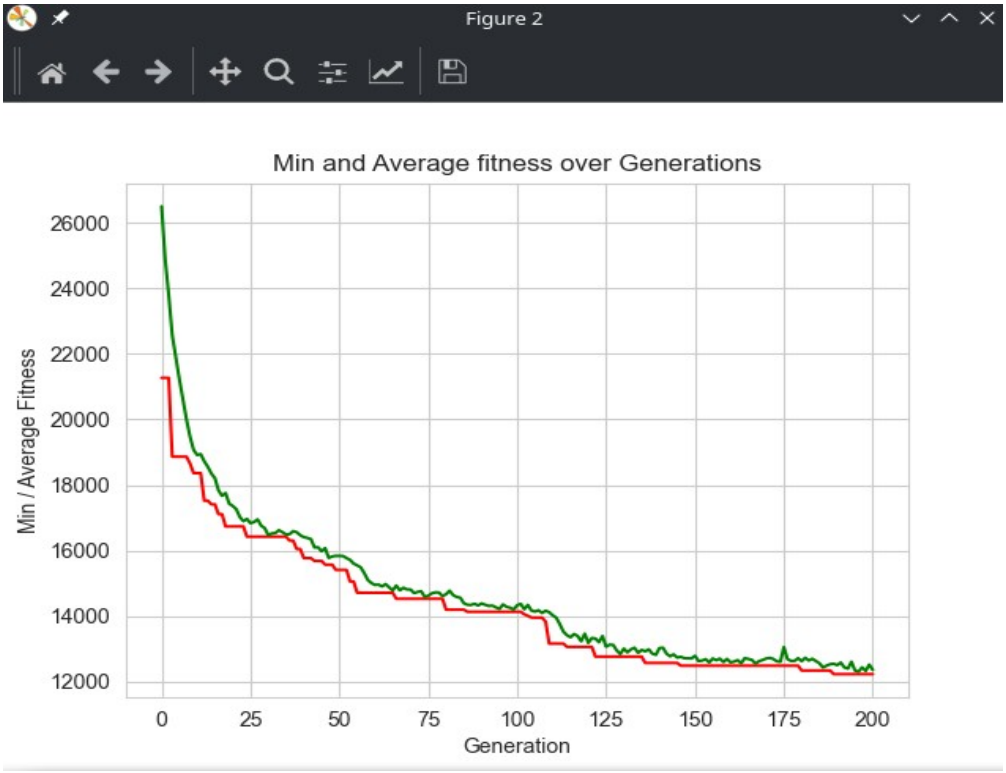
Možemo uočiti da su oko 90. generacije prosečna i minimalna putanja veoma blizu jedna drugoj, što znači da je algoritam konvergirao na „*optimalno*“ rešenje. Moramo uzeti u obzir da „*optimalno*“ može biti lokalni optimum, a ne globalni optimum.



Fitness 1: POPULATION_SIZE=100, P_CROSSOVER=0.9, P_MUTATUION=0.1

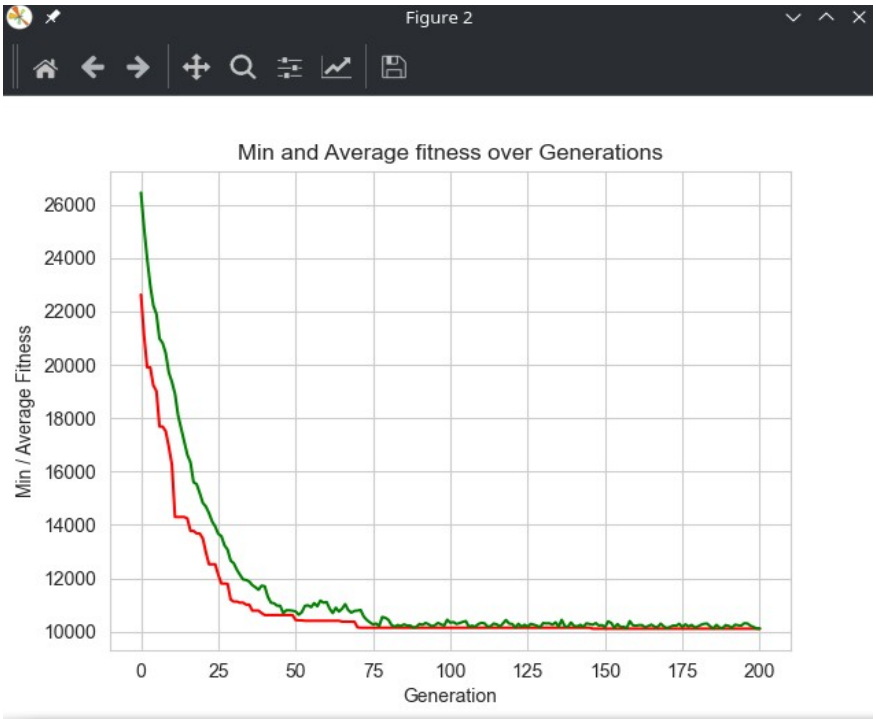
Genetski algoritam pronalazi lokalni optimizam u otprilike 90. iteraciji, mogli bismo zaustaviti genetski tok u tom trenutku kada je srednja Fitness vrednost blizu minimalne Fitness vrednosti sa nekim proizvoljnim procentom odstupanja, jer to znači da se cela populacija konvergirala oko lokalnog optimuma, a operater mutacije neće moći da generiše više različitih pojedinaca koji bi mogli da dovedu do istraživanja u drugim perspektivnijim regionima problematičnog prostora. Ako prestanemo ranije, mogli bismo da uštedimo resurse za obradu koji se troše na nepotrebne iteracije, ali ako imamo veći problem od trenutnog (znatno veći broj gradova tj. koordinata), može trebati znatno više generacija da bi se došlo do optimalnog rešenja.

Smanjenjem P_CROSSOVER parametra sa 0.9 na 0.1, do lokalnog optimuma dolazimo tek blizu 200. generacije, što je znatno lošije rešenje od prethodno ponuđenog.



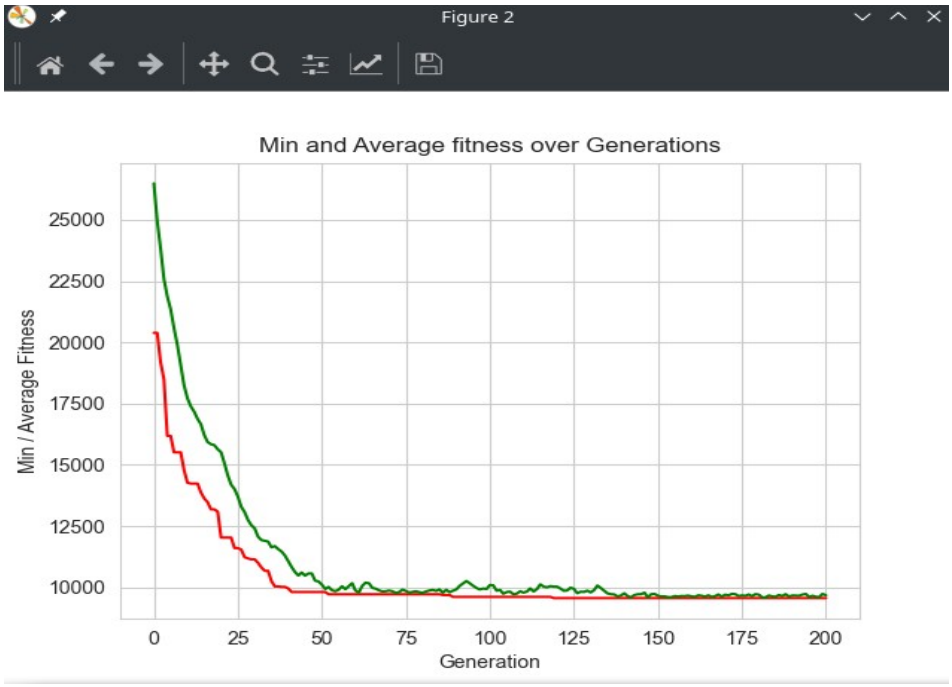
Fitness 2: POPULATION_SIZE=100, P_CROSSOVER=0.1, P_MUTATUION=0.1

U slučaju da P_CROSSOVER ostane na zadatoj vrednosti od 0.9, a P_MUTATUION smanjimo sa 0.1 na 0.05 dobijamo bolje rešenje, gde do lokalnog optimuma dolazimo oko 80. iteracije.

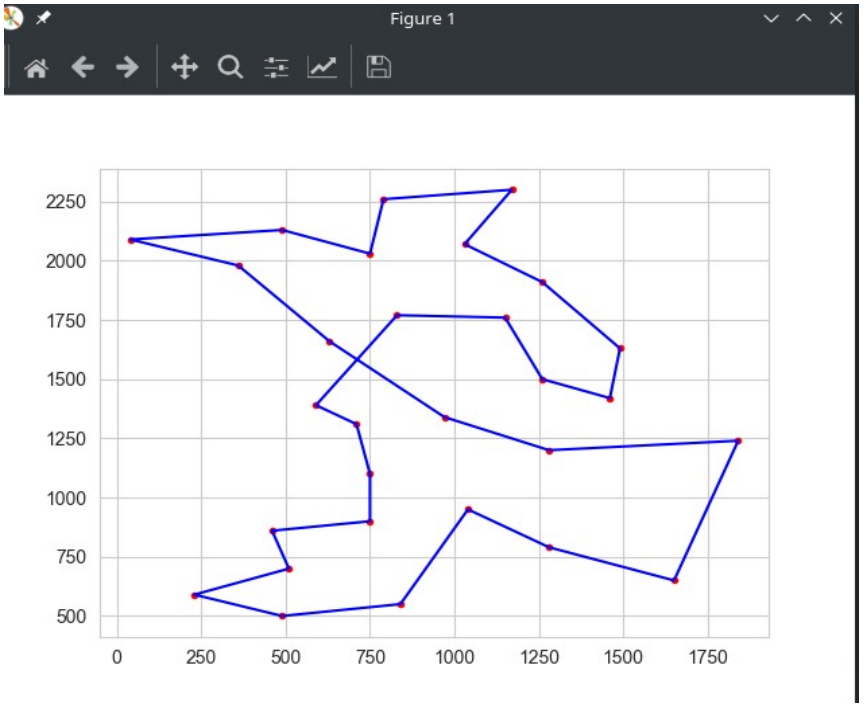


Fitness 3: POPULATION_SIZE=100, P_CROSSOVER=0.9, P_MUTATUION=0.05

Zarad daljeg poboljšanja našeg rešenja, pri P_CROSSOVER=0.9 I P_MUTATUION=0.05, dupliraćemo veličinu populacije po generaciji, dakle POPULATION_SIZE=200. Ovom izmenom dobijamo znatno brže postizanje lokalnog optimuma već oko 65. generacije, pri čemu smo takođe dobili niže minimalne vrednosti izabrane putanje, što se takođe može zaključiti po obliku najbolje putanje, koja je izmenjena u odnosu na prvi slučaj. Daljim dupliranjem POPULATION_SIZE=400 nismo dobili brže konvergiranje u smislu generacija, ali jesmo dobili još niže minimalne vrednosti putanje.



Fitness 4: POPULATION_SIZE=200, P_CROSSOVER=0.9, P_MUTATUION=0.05



Putanja 2: POPULATION_SIZE=200, P_CROSSOVER=0.9, P_MUTATUION=0.05