Imlementarea algoritmilor genetici

Asist.univ.dr Mihai Tudor

Obiective

- Înțelegerea principiilor de bază ale algoritmilor genetici (AG).
- Implementarea unui algoritm genetic simplu în Python.
- Aplicarea lui pe exemple clasice (maximizare funcție, problema șirului țintă).

1 Componentele de bază:

- Populația: un set de soluții candidate (cromozomi).
- Funcția de fitness: evaluează calitatea fiecărei soluții
- Selecția: alege indivizii cei mai apți pentru reproducere.
- Încrucișarea (crossover): combină doi părinți pentru a genera descendenți.
- Mutaţia: modifică aleator o parte din descendenți pentru diversitate.
- Criteriul de oprire: atingerea unui număr maxim de generații sau a unei soluții suficient de bune.

2 Structura de bază a unui algoritm genetic:

1. Inițializează populația P cu soluții aleatoare;
2. Evaluează fitness-ul fiecărui individ din populaţia P;
3. Cât timp nu s-a atins criteriul de oprire:
   (a) Selectează părinți din populaţia P;
   (b) Aplică încrucișarea pentru a obține descendenţi;
   (c) Aplică mutația asupra descendenţilor;
   (d) Evaluează fitness-ul descendenţilor;
   (e) Creează noua populație P';
4. Returnează cea mai bună soluție găsită;

3 Aplicaţii

Vom implementa o serie de algoritmi genetici în Python.

3.1 Maximizarea numărului de biți activi (biţi cu valoarea 1) într-un cromozom binar.

Vom implementa un algoritm genetic elementar având ca obiectiv optimizarea unui cromozom binar. Funcția de fitness este definită ca numărul total de biți de valoare 1 din cromozom. Problema de optimizare urmărește **maximizarea numărului de biți activați (1)** într-un individ. Un cromozom cu un număr mai mare de biți 1 este considerat superior din punct de vedere evolutiv. Pe parcursul generațiilor, algoritmul trebuie să conducă populația către indivizi cu fitness maxim.

Implementare

```
from random import*

# Functia de fitness (numarul de biti 1)
def fitness(cromozom):
    return sum(cromozom)

# Generarea populatie de cromozomi binari
def init_populatie(n, dim):
#n - dimensiunea populatiei
#dim - numarul de biti din cromozom
    populatie = []
    for i in range(n):
        cromozom = []
        for j in range(dim):
            bit = randint(0, 1)
            cromozom.append(bit)
        populatie.append(cromozom)
    return populatie

# Implementarea selectiei (folosim selectia de tip turneu)
def selectie(populatie, fitness):
# alegem doi cromozomi aleator din populatie
    a, b = sample(range(len(populatie)), 2)
# ajunge sa fie selectat cromozomul cu fitness-ul mai mare
    if fitness[a] > fitness[b]:
        return populatie[a]
    else:
        return populatie[b]

# Implementam one-point crossover
def incrucisare(p1, p2):
# generam random punctul de taiere
    p = randint(1, len(p1) - 1)
# returnam descendentii prin combinare de segmente de la parinti
    return p1[:p] + p2[p:], p2[:p] + p1[p:]
```

```
36
37 # Implementam mutatia simpla (flip bit)
38 def mutatie(cromozom, rata=0.01):
39     for i in range(len(cromozom)):
40         if random() < rata:
41             cromozom[i] = 1 - cromozom[i]
42
43 # Algoritmul genetic
44 def algoritm_genetic(dim_pop=10, dim_crom=8, generatii=20):
45 # pop - lista ce retine intreaga populatie de cromozomi
46     pop = init_populatie(dim_pop, dim_crom)
47
48     for g in range(generatii):
49 # fit - lista ce retine fitnnesu=ul fiecarui cromozom
50         fit = []
51         for ind in pop:
52           fit.append(fitness(ind))
53         noua_pop = []
54         while len(noua_pop) < dim_pop:
55             p1 = selectie(pop, fit)
56             p2 = selectie(pop, fit)
57             c1, c2 = incrucisare(p1, p2)
58             mutatie(c1)
59             mutatie(c2)
60             noua_pop.append(c1)
61             noua_pop.append(c2)
62
63         for i in range(dim_pop):
64             pop.append(noua_pop[i])
65         print("Generatia",g+1, "cel mai bun = ",max(fit))
66
67     return max(pop, key=fitness)
68
69 # Rulare
70 rezultat = algoritm_genetic()
71 print("Cea mai buna solutie gasita:", rezultat)
```

3.2 Maximizarea unei funcții numerice

Fie funcţia $f : \mathbb{R} \to \mathbb{R}, f(x) = x^2$. Creaţi un algoritm genetic care maximizează funcţia pe intervalul $[0, 31]$. Repezetarea cromozomilor se va fece la nivel binar pe 5 biţi.

Implementare

```
1
2 from random import*
3
4 # Functia de fitness (scorul fiecarui individ)
5 def fitness(cromozom):
6 # convertim cromozomul din sistem binar in sitem zecimal
7     x = 0
8     putere = 1
```

```
# parcurgem cromozomul de la dreapta la stanga
    for bit in reversed(cromozom):
        x=x+ bit * putere
        putere=putere* 2
#returnam cromozomul in sistem zecimal
    return x * x


# Generarea populatiei de cromozomi binari
def init_populatie(dim_pop, dim_cromozom):
    populatie = []
    for i in range(dim_pop):
        cromozom = []
        for j in range(dim_cromozom):
            cromozom.append(randint(0, 1))
        populatie.append(cromozom)
    return populatie

# Implementarea selectiei (folosim selectia de tip turneu)
def selectie(populatie, fitness):
# alegem doi cromozomi aleator din populatie
    a, b = sample(range(len(populatie)), 2)
# ajunge sa fie selectat cromozomul cu fitness-ul mai mare
    if fitness[a] > fitness[b]:
        return populatie[a]
    else:
        return populatie[b]

# Implementam one-point crossover
def incrucisare(p1, p2):
# generam random punctul de taiere
    p = randint(1, len(p1) - 1)
# returnam descendentii prin combinare de segmente de la parinti
    return p1[:p] + p2[p:], p2[:p] + p1[p:]


# Implementam mutatia simpla (flip bit)
def mutatie(cromozom, rata=0.01):
    for i in range(len(cromozom)):
        if random() < rata:
            cromozom[i] = 1 - cromozom[i]

# Algoritmul genetic
def algoritm_genetic(dim_pop=6, dim_crom=5, generatii=10):
# pop - lista ce retine intreaga populatie initiala de cromozomi

#Generam populatia initiala de cromozomi
    pop = init_populatie(dim_pop, dim_crom)

    for g in range(generatii):
# fit - lista ce retine fitnnesu=ul fiecarui cromozom
```

```
        fit = []
        for ind in pop:
          fit.append(fitness(ind))
#noua_pop - lista ce retine noua populatie la fiecare generatie
        noua_pop = []

        while len(noua_pop) < dim_pop: #criteriul de oprire
#aplicam selectia parintilor
            p1 = selectie(pop, fit)
            p2 = selectie(pop, fit)
#generam descendentii pentru noua populatie folosind one - point crossover                c1, c2 = incrucisare(p1, p2)
#aplicam mutatia asupra descendentilor pentru a crea diversitate
            mutatie(c1)
            mutatie(c2)
            noua_pop.append(c1)
            if len(noua_pop) < dim_pop:
                noua_pop.append(c2)
#dupa incheierea unui ciclu complet (o generatie), actualizam populatia initiala cu populatia noua generata
        pop = noua_pop
        print(f"Generatia {g+1}: cel mai bun fitness = {max(fit)}")

#returnam cel mai bun cromozom dupa incheierea tuturor ciclurilor de evolutie
    cel_mai_bun = pop[0]
    cel_mai_bun_fitness = fitness(cel_mai_bun)
    for cromozom in pop:
        f = fitness(cromozom)
        if f > cel_mai_bun_fitness:
            cel_mai_bun = cromozom
            cel_mai_bun_fitness = f

    return cel_mai_bun, cel_mai_bun_fitness

#Rulare
rezultat, fit = algoritm_genetic()
x = int("".join(map(str, rezultat)), 2)
print("\nCea mai buna solutie gasita:")
print("Cromozom:", rezultat)
print("x =", x)
print("Fitness =", fit)
```