

Laborator

Operatori Genetici

Asist.univ.dr Mihai Tudor

Obiective

- Implementarea mecanismului de selecție.
- Implementarea mecanismului de crossover.

1 Mecanismul de selecție

Selecția este procesul prin care cromozomii cei mai „buni” (cu fitness mai mare) sunt favorizați pentru a produce descendenți în generațiile următoare.

Rolul selecției:

- să păstreze cromozomii performanți;
- se menține și diversitatea populației, permițând uneori și indivizilor mai slabi să participe la procesul de selecție

Tipuri de selecție:

- **Roulette Wheel (selecția proporțională)** - Probabilitatea de a fi selectat este proporțională cu fitness-ul
 - Avantaje: Ușor de înțeles, menține diversitate;
 - Dezavantaje: Sensibil la valori extreme
- **Tournament** - Se alege aleator un număr mic de indivizi și se selectează cel mai bun
 - Avantaje: Simplu, controlabil;
 - Dezavantaje: Poate reduce diversitatea
- **Elitism** - Se păstrează automat cei mai buni indivizi din generație
 - Avantaje: Nu se pierd soluțiile bune
 - Dezavantaje: Poate duce la convergență rapidă

Implementare

În vederea implementării vom avea ca exemplu următoarea populație de cromozomi.

Cromozm	Codificare binară	Fitness	Probabilitate selecție
A	10101	25	$25 / (25 + 15 + 30 + 5) = 0.33$
B	01011	15	$15 / (25 + 15 + 30 + 5) = 0.20$
C	11100	30	$30 / (25 + 15 + 30 + 5) = 0.40$
D	00010	5	$5 / (25 + 15 + 30 + 5) = 0.07$

1. Selecția Roulette Wheel

- Se generează un număr aleator între 0 și suma totală a fitness-urilor.
- Se „parcurge” roata ruletei, adunând fitness-ul indivizilor până se depășește valoarea aleatorie.
- Individul ales este cel corespunzător aceluși segment.

```

1 from random import*
2
3 # Populatia si fitness-ul fiecarui individ
4 population = ['10101', '01011', '11100', '00010']
5 fitness_values = [25, 15, 30, 5]
6
7 # Calculam suma totala a fitness-urilor
8 total_fitness = sum(fitness_values)
9 print("Suma fitness-urilor:", total_fitness)
10
11 # Simulam selectia prin ruleta de 10 ori
12 selected = []
13
14 for i in range(10):
15     # Alegem un numar aleator intre 0 si totalul fitness-
16     # urilor
17     pick = uniform(0, total_fitness)
18     current = 0 #suma acumulata a fitness-urilor, creste pas
19     # cu pas.
20     # Parcurgem populatia si adunam fitness-urile
21     for j in range(len(population)):
22         current += fitness_values[j]
23         if current > pick:
24             selected.append(population[j])
25             break # oprim bucla individul a fost selectat
26
27 print("Indivizi selectati (prin ruleta):")
28 print(selected)

```

2. Selecția Turneu (Tournament)

- Se aleg aleator k indivizi (de exemplu 3).
- Se selectează cel cu cel mai mare fitness.
- Dacă k crește, selecția devine mai agresivă (favorizează cei mai buni).

```
1 from random import*
2
3 population = ['10101', '01011', '11100', '00010']
4 fitness_values = [25, 15, 30, 5]
5
6 selected = []
7 k = 3 # dimensiunea turneului
8
9 for i in range(10):
10     # Alegem aleator k indivizi din populatie
11     chosen = [] # lista celor alesi pentru turneu
12     while len(chosen) < k:
13         index = randint(0, len(population) - 1) # alegem o
14             pozitie aleatorie
15         if index not in chosen: # ne asiguram ca nu alegem
16             acelasi individ de doua ori
17             chosen.append(index)
18
19     # Determinam care dintre cei alesi are cel mai mare
20     fitness
21     best = chosen[0]
22     for i in chosen:
23         if fitness_values[i] > fitness_values[best]:
24             best = i
25
26     selected.append(population[best])
27
28 print("Indivizi selectati (prin turneu):")
29 print(selected)
```

Aplicații

1. Implementați selecția „ruletă” astfel încât să returneze o nouă populație de aceeași dimensiune.
2. Experimentați cu diferite valori ale lui k în selecția turneu (ex: $k=2, 5$). Ce concluzie obținem?
3. Exercițiul 3: Comparați distribuția indivizilor selectați pentru cele două metode — care favorizează mai mult indivizii buni?

2 Mecanismul de crossover

Crossover-ul combină informația genetică a doi cromozomi pentru a produce unul sau mai mulți descendenți.

Scopul mecanismului de crossover: exploatare — păstrăm trăsăturile bune de la cromozomii părinți și creăm variante noi pentru a explora cât mai multe regiuni din spațiul soluțiilor optime.

Tipuri de crossover:

- **One-point crossover** (pentru cromozomi binari) - Se alege un punct de tăiere pe cromozomi. Părțile din stânga punctului vin de la un părinte, cele din dreapta de la celălalt.
 - Avantaje: păstrează structuri (subșiruri) din cromozomii părinți;
 - Dezavantaje: poate rupe combinații bune dacă punctul e ales prost.
- **Two-point crossover** (pentru cromozomi binari)-Se aleg două puncte; segmentul dintre ele se schimbă între cromozomii părinți.
 - Avantaje: Combină mai mult material genetic; mai variat decât one-point.
 - Dezavantaje: Poate rupe structuri mari utile.
- **Uniform crossover** (pentru cromozomi binari) - Fiecare genă este aleasă aleator din părintele 1 sau părintele 2 .
 - Avantaje: Produce cromozomi copii foarte diverși; nu depinde de poziția genelor.
 - Dezavantaje: Poate distruge complet structuri bune.
- **Arithmetic crossover** (pentru cromozomi numerici) - cromozomul copil este o combinație liniară convexă între părinți: $child = \alpha \cdot 1 + (1 - \alpha) \cdot 2$, unde $\alpha \in [0, 1]$.
 - Avantaje: potrivit pentru optimizare continuă; menține valori medii.
 - Dezavantaje: nu păstrează valori exacte din părinți; risc de convergență prematură.

Implementare

1. One-point crossover

```
1 from random import *
2
3 parent1 = "1011001"
4 parent2 = "0100110"
5
6 # alegem un punct de crossover (intre 1 si len-1)
7 point = randint(1, len(parent1)-1)
8
9 # cromozomii "copii" se formeaza prin combinarea segmentelor
10 child1 = parent1[:point] + parent2[point:]
```

```

11 child2 = parent2[:point] + parent1[point:]
12
13 print("p1", parent1)
14 print("p2", parent2)
15 print("point", point)
16 print("c1", child1)
17 print("c2", child2)

```

2. Two-point crossover

```

1 from random import*
2
3 p1 = "1011001"
4 p2 = "0100110"
5 a = randint(0, len(p1)-2)
6 b = randint(a+1, len(p1)-1)
7
8 c1 = p1[:a] + p2[a:b] + p1[b:]
9 c2 = p2[:a] + p1[a:b] + p2[b:]
10
11 print("a,b:", a, b)
12 print("c1", c1)
13 print("c2", c2)

```

3. Uniform crossover

```

1 from random import randint
2
3 p1 = "1011001"
4 p2 = "0100110"
5 child = ""
6
7 for i in range(len(p1)):
8     if random() < 0.5:
9         child += p1[i]
10    else:
11        child += p2[i]
12
13 print("p1", p1)
14 print("p2", p2)
15 print("child", child)

```

4. Arithmetic crossover

```

1 from random import*
2
3 p1 = [1.0, 2.0, 3.0]
4 p2 = [4.0, 5.0, 6.0]
5
6 alpha = random.random() # generam alpha intre 0 si 1
7 child = []
8

```

```
9 for i in range(len(p1)):
10     child.append(alpha * p1[i] + (1 - alpha) * p2[i])
11
12 print("alpha:", alpha)
13 print("child:", child)
```