



UNIVERSITATEA BABEȘ-BOLYAI  
Facultatea de Matematică și Informatică



# INTELIGENȚĂ ARTIFICIALĂ

**Rezolvarea problemelor de căutare**

Strategii de căutare informată  
algoritmi evolutivi

Laura Dioșan

# Sumar

---

## A. Scurtă introducere în Inteligența Artificială (IA)

## B. Rezolvarea problemelor prin căutare

- Definirea problemelor de căutare
- Strategii de căutare
  - Strategii de căutare neinformate
  - Strategii de căutare informate
  - Strategii de căutare locale (Hill Climbing, Simulated Annealing, Tabu Search, Algoritmi evolutivi, PSO, ACO)
  - Strategii de căutare adversială

## C. Sisteme inteligente

- Sisteme care învață singure
  - Arbori de decizie
  - Rețele neuronale artificiale
  - Mașini cu suport vectorial
  - Algoritmi evolutivi
- Sisteme bazate pe reguli
- Sisteme hibride

# Materiale de citit și legături utile

---

- ❑ capitolul 14 din *C. Groșan, A. Abraham, Intelligent Systems: A Modern Approach, Springer, 2011*
- ❑ *M. Mitchell, An Introduction to Genetic Algorithms, MIT Press, 1998*
- ❑ capitolul 7.6 din *A. A. Hopgood, Intelligent Systems for Engineers and Scientists, CRC Press, 2001*
- ❑ Capitolul 9 din *T. M. Mitchell, Machine Learning, McGraw-Hill Science, 1997*

# Căutare locală

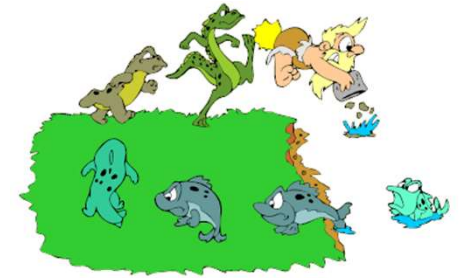


## □ Tipologie

- Căutare locală simplă - se reține o singură stare vecină
  - Hill climbing → alege cel mai bun vecin
  - Simulated annealing → alege probabilistic cel mai bun vecin
  - Căutare tabu → reține lista soluțiilor recent vizitate
- Căutare locală în fascicol (beam local search) – se rețin mai multe stări (o populație de stări)
  - Algoritmi evolutivi
  - Optimizare bazată pe comportamentul de grup (Particle swarm optimisation)
  - Optimizare bazată pe furnici (Ant colony optimisation)

# Algoritmi evolutivi

---



## □ Tipuri de algoritmi evolutivi

- Algoritmi genetici
- Strategii evolutive
- Programare evolutivă
- Programare genetică

# Algoritmi genetici

---



- ❑ Aspecte teoretice
- ❑ Algoritm
  - Schema generală a unui AGS
  - Reprezentare și operatori
- ❑ Exemplu
- ❑ Proprietăți
- ❑ Aplicații



# Algoritmi genetici – aspecte teoretice

- Propuși
  - J. Holland → AG simpli (AGS)
- Căutare
  - **Concurențială**, ghidată de calitatea **absolută** a indivizilor
- Operatori de căutare
  - Selecția
  - Încrucișarea **ȘI** mutația
- Elemente speciale
  - Accent deosebit pe încrucișare



# Algoritmi genetici – schema generală

## Algoritm generațional

Inițializare  $P(0)$

Evaluare( $P(0)$ )

$g = 0$ ;

*while* (*not* condiție\_stop) *do*

*repeat*

        Selectarea a 2 părinți  $p_1$  și  $p_2$  din  $P(g)$

        Încrucișare( $p_1, p_2$ )  $\rightarrow o_1$  și  $o_2$

        Mutație( $o_1$ )  $\rightarrow o_1^*$

        Mutație( $o_2$ )  $\rightarrow o_2^*$

        Evaluare( $o_1^*$ )

        Evaluare( $o_2^*$ )

        Adăugare  $o_1^*$  și  $o_2^*$  în  $P(g+1)$

*until*  $P(g+1)$  este plină

$g++$

*endWhile*

## Algoritm steady-state

Inițializare  $P$

Evaluare( $P$ )

*while* (*not* condiție\_stop) *do*

*For*  $i = 1$  *to*  $|P|$

        Selectarea a 2 părinți  $p_1$  și  $p_2$  din  $P(g)$

        Încrucișare( $p_1, p_2$ )  $\rightarrow o_1$  și  $o_2$

        Mutație( $o_1$ )  $\rightarrow o_1^*$

        Mutație( $o_2$ )  $\rightarrow o_2^*$

        Evaluare( $o_1^*$ )

        Evaluare( $o_2^*$ )

$B = \text{Best}(o_1^*, o_2^*)$

$W = \text{Worst}(o_1^*, o_2^*)$

        Dacă  $B$  e mai bun ca  $W$ ,  $W \leftarrow B$

*EndFor*

*endWhile*





# Algoritmi genetici – schema generală

## Algoritm generațional

1. Generarea aleatoare a unei populații (generația 0) cu  $n$  cromozomi
2. Evaluarea tuturor cromozomilor
3. Crearea unei noi populații (generații) prin repetarea următorilor 4 pași
  - Selecția, bazată pe fitness, a 2 părinți
  - Încrucișarea părinților pentru obținerea unui descendent cu o anumită probabilitate; dacă încrucișarea nu are loc, descendentul va fi:
    - Unul dintre părinți
    - Cel mai bun dintre părinți
  - Mutația cu o anumită probabilitate a fiecărui element al descendentului
  - Acceptarea descendentului și plasarea lui în noua populație (generație)
4. Înlocuirea vechii populații cu noua populație (schimbul de generații)
5. Testarea condițiilor de terminare a căutării; dacă ele sunt satisfăcute, se returnează cea mai bună soluție din populația (generația) curentă
6. Ciclarea algoritmului – întoarcerea la pasul 2



# Algoritmi genetici – schema generală

## Algoritm steady-state

1. Generarea aleatoare a unei populații cu  $n$  cromozomi
2. Evaluarea tuturor cromozomilor
3. Crearea unei noi populații prin repetarea următorilor 4 pași
  - Selecția, bazată pe fitness, a 2 părinți
  - Încrucișarea părinților pentru obținerea unui descendent cu o anumită probabilitate; dacă încrucișarea nu are loc, descendentul va fi:
    - Unul dintre părinți
    - Cel mai bun dintre părinți
  - Mutația cu o anumită probabilitate a fiecărui element al descendentului
  - Alegerea celui mai bun descendent B
  - Dacă B este mai bun decât cel mai slab individ al populației W, atunci B îl înlocuiește pe W
4. Testarea condițiilor de terminare a căutării; dacă ele sunt satisfăcute, se returnează cea mai bună soluție din populația (generația) curentă
5. Ciclarea algoritmului – întoarcerea la pasul 2

# Algoritmi genetici – reprezentare și operatori



- Reprezentare
  - Stringuri binare (inițial), de numere întregi, de numere reale, de alte elemente
- Populația
  - $\mu$  părinți,  $\mu$  descendenți
- Selecția pentru recombinare
  - Propoțională cu fitness-ul
- Recombinarea
  - Cu  $n$  puncte de tăietură sau uniformă cu o probabilitate  $p_c$  fixată ce acționează la nivel de cromozom
- Mutația
  - *Bitwise bit-flipping* cu o probabilitate  $p_m$  fixată pentru fiecare genă (bit)
- Selecția pentru supraviețuire
  - Toți descendenții înlocuiesc părinții



# Algoritmi genetici – exemplu

- Să se determine valoarea maximă a funcției

$$f: \{0,1,\dots,31\} \rightarrow \mathbb{Z}, f(x) = x^2$$

- Configurarea AG

- Stringuri binare de lungime 5, ex.  $c=(10101) \rightarrow x=21$
- O populație cu  $\mu = 4$  cromozomi
- Selecție proporțională prin ruletă
- Încrucișare cu 1 punct de tăietură
- Mutație tare

- Evaluare  $\rightarrow$  optimizare prin maximizare



# Algoritmi genetici – exemplu

Inițializare

No cromo- zom	Cromo zom
1	01101
2	11000
3	01000
4	10011
sumă	



# Algoritmi genetici – exemplu

Evaluare

No cromozom	Cromozom	Valoarea $x$	Fitness $f(x^2)$
1	01101	13	169
2	11000	24	576
3	01000	8	64
4	10011	19	361
sumă			1170



# Algoritmi genetici – exemplu

## Selectie

No cromo- zom	Cromo zom	Valoar ea x	Fitness $f(x^2)$	$P_{selSP}(i)$	$\Sigma P_{selSP}(i)$
1	01101	13	169	$\frac{169}{1170} = 0.14$	0.14
2	11000	24	576	$\frac{576}{1170} = 0.49$	0.63
3	01000	8	64	$\frac{64}{1170} = 0.06$	0.69
4	10011	19	361	$\frac{361}{1170} = 0.31$	1.00
sumă			1170		



# Algoritmi genetici – exemplu

## Selectie

No cromozom	Cromozom	Valoarea x	Fitness	$P_{selSP}(i)$	$\Sigma P_{selSP}(i)$	$r_1=0.5$	$r_2=0.8$
1	01101	13	169	$\frac{169}{1170}=0.14$	0.14		
2	11000	24	576	$\frac{576}{1170}=0.49$	0.63	X	
3	01000	8	64	$\frac{64}{1170}=0.06$	0.69		
4	10011	19	361	$\frac{361}{1170}=0.31$	1.00		x
sumă			1170				

$p_1=c_2 = (11000)$  și  $p_2 = c_4 = (10011)$





# Algoritmi genetici – exemplu

## Încrucișare

No cromozom	Cromozomi părinți	Cromozomi fii	Valoarea x (pt. fii)	Fitness (pt. fii)
2	11000	11011	27	729
4	10011	10000	16	256

## Mutație

No cromozom	Cromozomi fii	Cromozomi fii*	Valoarea x (pt. fii*)	Fitness (pt. fii*)
o1	11011	10011	19	361
o2	10000	10010	18	324



# Algoritmi genetici – exemplu

Adăugarea în următoarea generație

No cromo- zom	Cromo zom
1	10011
2	10010
3	
4	



# Algoritmi genetici – exemplu

## Selectie

No cromozom	Cromozom	Valoarea $x$	Fitness $f(x^2)$	$P_{\text{selSP}}(i)$	$\Sigma P_{\text{selSP}}(i)$	$r_1=0.1$	$r_2=0.7$
1	01101	13	169	$\frac{169}{1170}=0.14$	0.14	x	
2	11000	24	576	$\frac{576}{1170}=0.49$	0.63		
3	01000	8	64	$\frac{64}{1170}=0.06$	0.69		
4	10011	19	361	$\frac{361}{1170}=0.31$	1.00		x
sumă			1170				



# Algoritmi genetici – exemplu

## Selectie

No cromozom	Cromozom	Valoarea $x$	Fitness $f(x^2)$	$P_{\text{selSP}}(i)$	$\Sigma P_{\text{selSP}}(i)$	$r_1=0.5$	$r_2=0.8$
1	01101	13	169	$\frac{169}{1170}=0.14$	0.14	x	
2	11000	24	576	$\frac{576}{1170}=0.49$	0.63		
3	01000	8	64	$\frac{64}{1170}=0.06$	0.69		
4	10011	19	361	$\frac{361}{1170}=0.31$	1.00		x
sumă			1170				

$p_1=c_1 = (01101)$  și  $p_2 = c_4 = (10011)$



# Algoritmi genetici – exemplu

## Încrucișare

No cromozom	Cromozomi părinți	Cromozomi fii	Valoarea x (pt. fii)	Fitness (pt. fii)
1	01101	01011	11	121
4	10011	10101	21	441

## Mutație

No cromozom	Cromozomi fii	Cromozomi fii*	Valoarea x (pt. fii*)	Fitness (pt. fii*)
o1	01011	00011	3	9
o2	10101	10111	23	529

# Algoritmi genetici – exemplu



Adăugarea în următoarea generație

No cromo- zom	Cromo zom
1	10011
2	10010
3	00011
4	10111

# Algoritmi genetici – proprietăți



- Cromozomi liniari de aceeași dimensiune
- Evidențiază avantajele combinării informațiilor de la părinți buni prin încrucișare
- Numeroase variante
- Numeroși operatori (selecție, încrucișare, mutație)
- Nu sunt foarte rapizi
- Euristici bune pentru probleme de combinatorică

# Algoritmi genetici – aplicații



- ❑ Probleme de combinatorică
- ❑ Optimizări în proiectarea compoziției materialelor și a formei aerodinamice a vehiculelor (auto, aeriene, navale, trenuri)
- ❑ Optimizări în proiectarea structurală și funcțională a clădirilor (locuințe, fabrici, etc)
- ❑ Optimizări în robotică
- ❑ Optimizări în proiectarea circuitelor digitale



# Strategii evolutive

---



- ❑ Aspecte teoretice
- ❑ Algoritm
  - Schema generală
  - Reprezentare și operatori
- ❑ Exemplu
- ❑ Proprietăți
- ❑ Aplicații



# Strategii evolutive – aspecte teoretice

- Propuse
  - În anii '60-'70 în Germania de către Bienert, Rechenberg și Schwefel
- Căutare
  - **Concurențială**, ghidată de calitatea **absolută** a indivizilor
- Operatori de căutare
  - Selecția
  - Încrucișarea **ȘI** mutația
- Elemente speciale
  - Auto-adaptarea parametrilor (în special a parametrilor mutației)



# Strategii evolutive – schema generală

Inițializare  $P(0)$

Evaluare( $P(0)$ )

$g = 0$ ;

*while* (*not* condiție\_stop) *do*

*repeat*

        Selectarea a 2 părinți  $p_1$  și  $p_2$  din  $P(g)$

        Încrucișare( $p_1, p_2$ )  $\rightarrow o_1$

        Mutație( $o_1, param$ )  $\rightarrow o_1^*, param'$

        Evaluare( $o_1^*$ )

        Adăugare  $o_1^*$  în  $P(g+1)$

*until*  $P(g+1)$  este plină

$g++$

*endWhile*

# Strategii evolutive – reprezentare și operatori

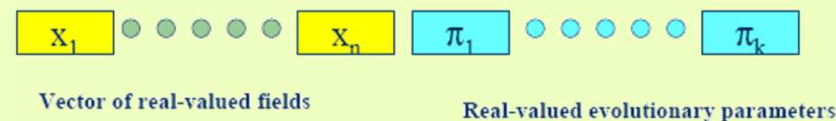


- Reprezentare
  - Reală
  - Codează și rata de mutație
- Populația
  - $\mu$  părinți,  $\lambda$  descendenți
- Selecția pentru recombinare
  - Uniformă aleatoare
- Recombinarea
  - Discretă sau intermediară
- Mutația
  - Perturbare Gaussiană
  - Auto-adaptare a pasului de mutație
- Selecția pentru supraviețuire
  - $(\mu, \lambda)$  sau  $(\mu + \lambda)$

# Strategii evolutive – reprezentare și operatori



- Pp. că dorim minimizarea funcției  $f: R^n \rightarrow R$
- Reprezentare
  - 3 părți:
    - Variabile obiect:  $x_1, x_2, \dots, x_n$  cu  $x_i \in R \rightarrow$  reprezentare reală
    - Parametri posibili ai SE:
      - Pași de mutație:  $\sigma_1, \dots, \sigma_{n(\sigma)}$
      - Unghiuri de rotație  $\omega_1, \dots, \omega_{n(\alpha)}$
  - Completă
    - $n(\sigma)=n, n(\alpha) = n(n-1)/2$  – nr de perechi  $(i,j), i, j = 1, 2, \dots, n$



# Strategii evolutive – reprezentare și operatori

---



- Selecția părinților (pentru reproducere)
  - Uniformă aleatoare
  - Fiecare individ are aceeași probabilitate de a fi selectat

# Strategii evolutive – reprezentare și operatori



## □ Reproducerea

- Combină doi sau mai mulți părinți
- Crează un singur descendent

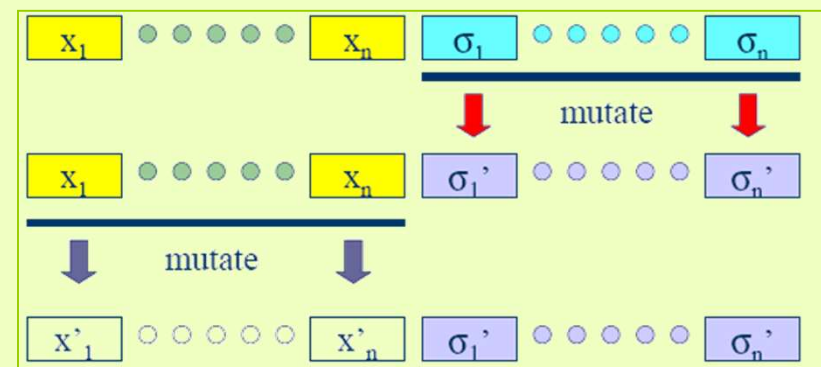
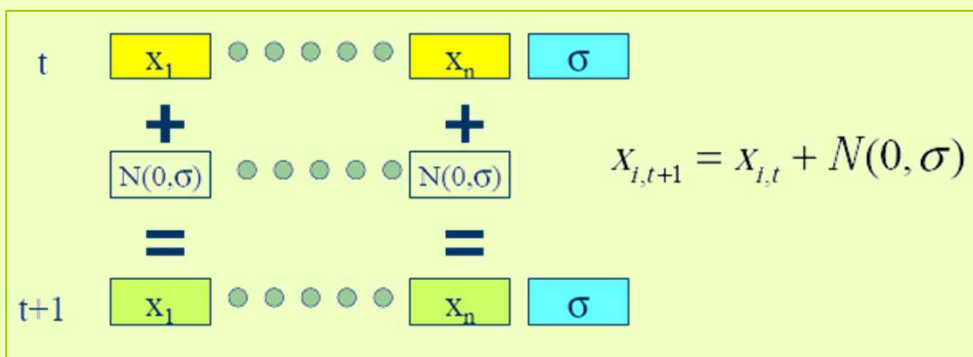
	2 părinți	Câte 2 părinți pentru fiecare element $x_i$ al unui cromozom
$z_i = (x_i + y_i) / 2$	Intermediară locală	Intermediară globală
$z_i$ este fie $x_i$ , fie $y_i$ (alegerea fiind aleatoare)	Discretă locală	Discretă globală

# Strategii evolutive – reprezentare și operatori



## □ Mutația

- Parametrii  $\sigma$  se coevoluează cu soluția  $x$
- Mutație Gaussiană
  - $\sigma$  este evoluat în  $\sigma'$
  - $x'_i = x_i + N(0, \sigma')$
- Nu este necesar ca parametrii  $\sigma$  să evolueze (să se modifice) cu aceeași frecvență ca soluția ( $x$ )
- Noua pereche  $(x', \sigma')$  se evaluează de 2 ori:
  - $x'$  este bună dacă evaluarea  $f(x')$  este bună
  - $\sigma'$  este bun dacă  $x'$  este bună





# Strategii evolutive – reprezentare și operatori



□ Cum este evoluat pasul de mutație din  $\sigma$  în  $\sigma'$  ? → diferite metode

## ■ Regula succesului 1/5

- Se determină procentajul  $p_s$  al mutațiilor “folositoare” (care au îmbunătățit potențiala soluție) din ultimele  $k$  iterații
- Se modifică  $\sigma$  după fiecare  $k$  iterații astfel:
  - $\sigma = \sigma / c$ , dacă  $p_s > 1/5$
  - $\sigma = \sigma * c$ , dacă  $p_s < 1/5$
  - $\sigma = \sigma$ , dacă  $p_s = 1/5$ ,
- unde  $0.8 \leq c \leq 1$

## ■ Regula auto-adaptării

- Mutație necorelată cu un singur parametru  $\sigma$
- Mutație necorelată cu  $n$  parametri  $\sigma$
- Mutație corelată

# Strategii evolutive – reprezentare și operatori

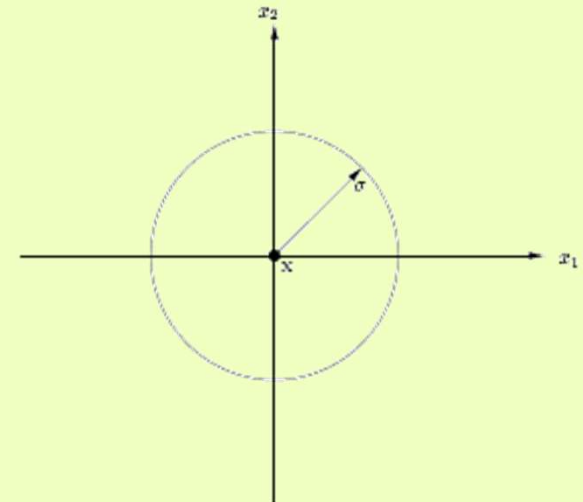


## □ Mutație necorelată cu un singur parametru $\sigma$

■ Cromozomi de forma:  $(x_1, x_2, \dots, x_n, \sigma)$

### ■ Mutație

- $\sigma' = \sigma * \exp(\tau * N(0, 1))$
- $x_i' = x_i + \sigma' * N(0, 1)$
- Unde  $\tau$  - rata de învățare
  - de obicei  $\tau = 1/(n^{1/2})$
- Dacă  $\sigma' < \varepsilon_0 \rightarrow \sigma' = \varepsilon_0$



# Strategii evolutive – reprezentare și operatori



## □ Mutație necorelată cu $n$ parametri $\sigma$

- Cromozomi de forma:  $(x_1, x_2, \dots, x_n, \sigma_1, \sigma_2, \dots, \sigma_n)$

- Mutație

- $\sigma'_i = \sigma_i * \exp(\tau' * N(0,1) + \tau * N_i(0,1))$

- $x'_i = x_i + \sigma'_i * N_i(0,1)$

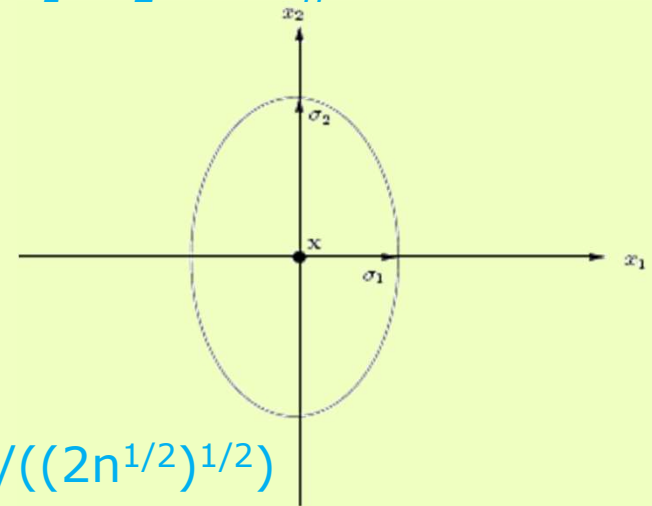
- unde:

- $\tau'$  - rata globală de învățare

- $\tau$  - rata individuală de învățare

- de obicei  $\tau' = 1/((2n)^{1/2})$  și  $\tau = 1/((2n^{1/2})^{1/2})$

- dacă  $\sigma' < \varepsilon_0 \rightarrow \sigma' = \varepsilon_0$



# Strategii evolutive – reprezentare și operatori



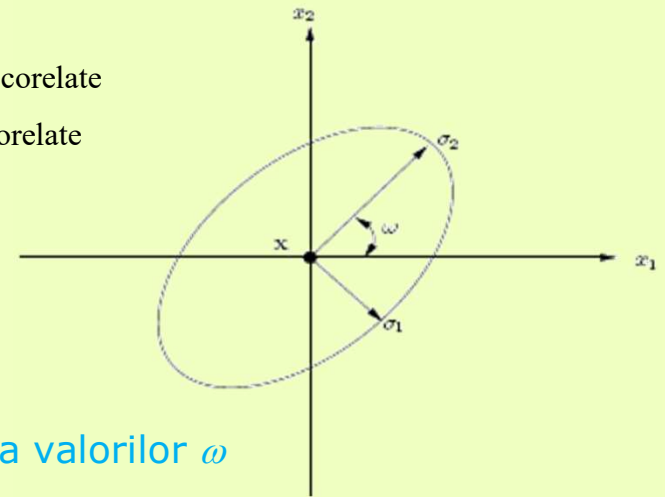
## □ Mutație corelată cu $n+k$ parametri

- Cromozomi de forma:  $(x_1, x_2, \dots, x_n, \sigma_1, \sigma_2, \dots, \sigma_n, \omega_1, \omega_2, \dots, \omega_k)$ ,
  - unde  $k=n(n-1)/2$
  - Matricea de covariație  $\mathbf{C}$  este definită prin:

$$c_{ij} = \begin{cases} \sigma_i^2, & \text{dacă } i = j \\ 0, & \text{dacă } i \text{ și } j \text{ nu sunt corelate} \\ \frac{1}{2}(\sigma_i^2 - \sigma_j^2) * \tan(2\omega_{ij}), & \text{dacă } i \text{ și } j \text{ sunt corelate} \end{cases}$$

## ■ Mutație

- $\sigma'_i = \sigma_i * \exp(\tau' * N(0,1) + \tau * N_i(0,1))$
- $\omega'_{ij} = \omega_{ij} + \beta * N(0,1)$
- $\mathbf{x}' = \mathbf{x} + \mathbf{N}(0, \mathbf{C}')$
- unde:
  - $\mathbf{x} = (x_1, x_2, \dots, x_n)$
  - $\mathbf{C}'$  – matricea de covariație  $\mathbf{C}$  după mutarea valorilor  $\omega$
  - $\tau'$  - rata globală de învățare
  - $\tau$  - rata inteligentă de învățare
  - de obicei  $\tau' = 1/((2n)^{1/2})$  și  $\tau = 1/((2n^{1/2})^{1/2})$  și  $\beta \approx 5^\circ$
- dacă  $\sigma' < \varepsilon_0 \rightarrow \sigma' = \varepsilon_0$
- dacă  $|\omega'_{ij}| > \pi \rightarrow \omega'_{ij} = \omega'_{ij} - 2\pi \text{ sign}(\omega'_{ij})$



# Strategii evolutive – reprezentare și operatori



## □ Selecția de supraviețuire

- Aplicată după crearea a  $\lambda$  descendenți din  $\mu$  părinți prin recombinare și mutație
- Alegerea celor mai buni  $\mu$  indivizi din
  - Mulțimea copiilor – SE ( $\mu, \lambda$ )
    - Selecție “uitucă”
    - Are performanțe mai bune
  - Mulțimea părinților și copiilor – SE( $\mu + \lambda$ )
    - Selecție elitistă
- De obicei,  $\lambda = 7 * \mu$  (→ presiune de selecție mare)

# Strategii evolutive – proprietăți



## □ Caracteristici

- cromozomi liniari de aceeași dimensiune
- oferă viteză de lucru
- lucrează cu vectori de numere reale
- se bazează pe o teorie matematică fundamentată
- evoluează și parametrii algoritmului în sine (auto-adaptează parametrii mutației)

## □ SE inițiale

- $SE(\mu+\lambda)$ , cu  $\mu=1$ ,  $\lambda=1$
- Căutare locală de tip Hill Climbing
- Dar, cromozomul codează și:
  - rata de mutație
  - strategie de modificare pentru deviația standard a distribuției mutației

# Strategii evolutive - aplicații



- ❑ Probleme de optimizare numerică
- ❑ Optimizarea formeii lentilelor necesare refracției luminii
- ❑ Distribuția lichidului într-o rețea sangvină
- ❑ Curba Brachystochrone
- ❑ Rezolvarea cubului Rubik

# Programare evolutivă

---



- Aspecte teoretice
- Algoritm
  - Schema generală
  - Reprezentare și operatori
- Proprietăți
- Aplicații



# Programare evolutivă – aspecte teoretice



- Propusă
  - în SUA în anii 1960 de către D. Fogel
- Căutare
  - **Concurențială**, ghidată de calitatea **relativă** a indivizilor → selecția de supraviețuire
- Operatori de căutare
  - Selecția
  - **DOAR** mutația
- Elemente speciale
  - AE fără recombinare
  - Auto-adaptarea parametrilor (similar SE)

# Programare evolutivă – schema generală



Inițializare  $P(0)$

Evaluare( $P(0)$ )

$g = 0;$

*while* (*not* condiție\_stop) *do*

    Pentru fiecare cromozom  $c_i$  din  $P(g)$

        Mutație( $c_i, param$ )  $\rightarrow o_i, param'$

        Evaluare( $o_i$ )

    Alegerea probabilistică a  $\mu$  cromozomi dintre  $c_1, \dots, c_\mu, o_1, \dots, o_\mu$  și  
    adăugarea lor în  $P(g+1)$

$g++$

*endWhile*

# Programare evolutivă – reprezentare și operatori



- Reprezentare
  - Reală
  - Codează și parametrii mutației (pasul de mutație)
- Populația
  - $\mu$  părinți,  $\lambda = \mu$  descendenți
- Selecția pentru mutație
  - Deterministă
- Mutația
  - Perturbare Gaussiană
  - Auto-adaptare a parametrilor
- Selecția pentru supraviețuire
  - $(\mu+\mu)$  probabilistică

# Programare evolutivă – reprezentare și operatori



□ Pp că dorim optimizarea funcției  $f: R^n \rightarrow R$

□ Reprezentarea cromozomilor

■ 2 părți:

□ Variabile obiect:  $x_1, x_2, \dots, x_n$

□ Pași de mutație:  $\sigma_1, \dots, \sigma_n$

■ Completă

□  $(x_1, \dots, x_n, \sigma_1, \dots, \sigma_n)$

# Programare evolutivă – reprezentare și operatori

---



- Selecția părinților (pentru mutație)
  - Fiecare părinte produce prin mutație un descendent → selecție
    - deterministă
    - ne-bazată pe calitatea (fitnessul) indivizilor

# Programare evolutivă – reprezentare și operatori



## □ Mutația

- Singurul operator care introduce variație în PE
- Cromozomul  $(x_1, \dots, x_n, \sigma_1, \dots, \sigma_n)$
- Modificări de tip Gaussian
  - $\sigma'_i = \sigma_i * (1 + \alpha * N_i(0, 1))$
  - $x'_i = x_i + \sigma'_i * N_i(0, 1)$
  - $\alpha \approx 0.2$  – rata de învățare
- Limitări
  - dacă  $\sigma' < \varepsilon_0 \rightarrow \sigma' = \varepsilon_0$

# Programare evolutivă – reprezentare și operatori



## □ Selecția de supraviețuire

- Populația (la momentul  $t$ ) are  $\mu$  părinți care produc  $\mu$  descendenți

- Campionat “fiecare cu fiecare” (*round-robin*)

- Fiecare soluție  $s_i$ ,  $i = 1, 2, \dots, \mu^2$  din cei  $\mu$  părinți și  $\mu$  descendenți este comparată cu alte  $q$  soluții (diferite de  $s$ ) alese aleator (din aceeași mulțime a părinților și urmașilor)
- Pentru fiecare soluție  $s_i$  se stabilește de câte ori a câștigat un meci jucat

$$p_i = \sum_{l=1}^q p_{il},$$

$$p_{il} = \begin{cases} 1, & f(s_i) \text{ e mai bun ca } f(s_l) \\ 0, & \text{altfel} \end{cases}$$

- Se aleg cele mai bune  $\mu$  soluții (cu cele mai multe jocuri câștigate -  $p_i$ )

- Parametrul  $q$  reglează presiunea de selecție

- de obicei  $q = 10$

- ➔ procesul de căutare este ghidat de calitatea *relativă* a indivizilor



# Programare evolutivă – proprietăți

---

- Cromozomi liniari de aceeași dimensiune
- Algoritmi evolutivi fără recombinare
- Auto-adaptare a parametrilor (similar *SE*)
- Cadru foarte permisiv: orice reprezentare și mutație poate funcționa bine
  - Mutație uniformă
  - Mutație Cauchy
  - Mutație Lévy



# Programare evolutivă – aplicații



- Învățare automată cu mașini cu stări finite
- Optimizare numerică
- Distribuția și planificarea traficului în rețele
- Proiectarea farmaceutică
- Epidemiologie
- Detecția cancerului
- Planificare militară
- Procesarea semnalelor

# Programare genetică

---



- Aspecte teoretice
- Algoritm
  - Schema generală
  - Reprezentare și operatori
- Exemplu
- Aplicații

# Programare genetică – aspecte teoretice



## ■ Propusă

- În SUA în anii 1990 de către J. Koza
- Evoluarea de programe → evaluarea unui individ implică execuția programului codat în cromozom

## ■ Căutare

- **Concurențială**, ghidată de calitatea **absolută** a indivizilor

## ■ Operatori de căutare

- Selecția
- Recombinarea **SAU** mutația

## ■ Elemente speciale

- Cromozomi ne-liniari (arbori sau grafe) și de dimensiuni diferite
- Pot folosi mutația (dar nu e neapărat necesar)

# Programare genetică – schema generală



```
Inițializare  $P(0)$   
Evaluare( $P(0)$ )  
 $g = 0$ ;  
while (not condiție_stop) do  
    repeat  
        Selectarea a 2 părinți  $p_1$  și  $p_2$  din  $P(g)$   
        Încrucișare( $p_1, p_2$ )  $\rightarrow o_1$  și  $o_2$   
        Mutație( $o_1$ )  $\rightarrow o_1^*$   
        Mutație( $o_2$ )  $\rightarrow o_2^*$   
        Evaluare( $o_1^*$ )  
        Evaluare( $o_2^*$ )  
        Adăugare  $o_1^*$  și  $o_2^*$  în  $P(g+1)$   
    until  $P(g+1)$  este plină  
     $g++$   
endWhile
```

# Programarea genetică – reprezentare și operatori



- Reprezentare
  - Structuri arborescente de dimensiune variabilă
- Populația
  - $\mu$  părinți,  $\mu$  descendenți
- Selecția pentru recombinare
  - Propoțională cu fitness-ul
- Recombinarea
  - Schimbul de sub-arbori
- Mutația
  - Schimbări aleatoare în arbore
- Selecția pentru supraviețuire
  - Schema generațională - toți descendenții înlocuiesc părinții
  - Schema steady-state – cu elitism

# Programarea genetică – reprezentare și operatori



## □ Reprezentare

- Potențialele soluții sub forma unor arbori → implicații:

- Indivizi adaptivi

- Dimensiunea cromozomilor nu este prefixată
- Dimensiunea cromozomilor depinde de adâncimea și factorul de ramificare al arborilor

- Gramatici specifice domeniului problemei de rezolvat

- Necesitatea definirii exacte a unei gramatici reprezentative pentru problema abordată
- Gramatica trebuie să permită reprezentarea oricărei soluții posibile/potențiale

# Programarea genetică – reprezentare și operatori



## □ Reprezentare

### ■ Gramatica conține:

- Setul de terminale specifică toate variabilele și constantele problemei
- Setul de funcții conține toți operatorii care pot fi aplicați terminalelor:
  - Operatori aritmetici (+, -, \*, /, sin, cos, log, ...)
  - Operatori de tip Boolean (and, or, not, ...)
  - Operatori de tip instrucțiune (if-then, for, while, set, ...)
- Regulile care asigură obținerea unor soluții potențiale valide

### ■ De ex. arbori care codifică

- Formule logice
- Formule aritmetice
- Programe

# Programarea genetică – reprezentare și operatori

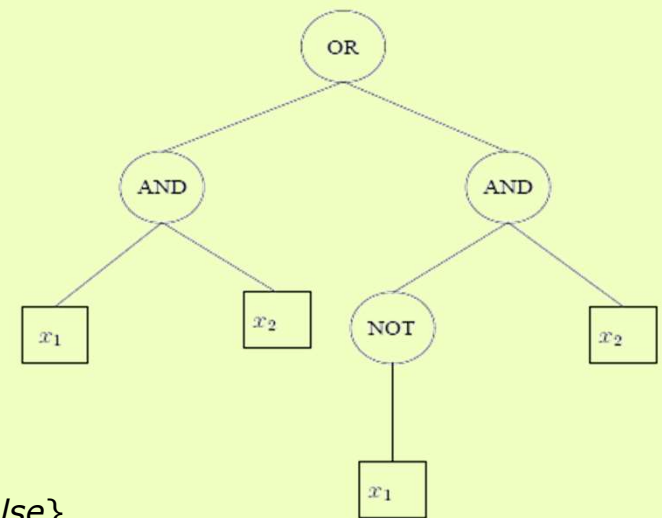


## □ Reprezentare

### ■ exemplu de evoluare a unei expresii logice

- Problemă: să se determine expresia logică identificată prin datele:

	x1	x2	Output
	0	0	0
	0	1	1
	1	0	1
	1	1	0



- Setul de funcții  $F = \{AND, OR, NOT\}$
- Setul de terminale  $T = \{x_1, x_2\}$ , cu  $x_1, x_2 \in \{True, False\}$
- Soluție:  $(x_1 AND NOT x_2) OR (NOT x_1 AND x_2)$



# Programarea genetică – reprezentare și operatori

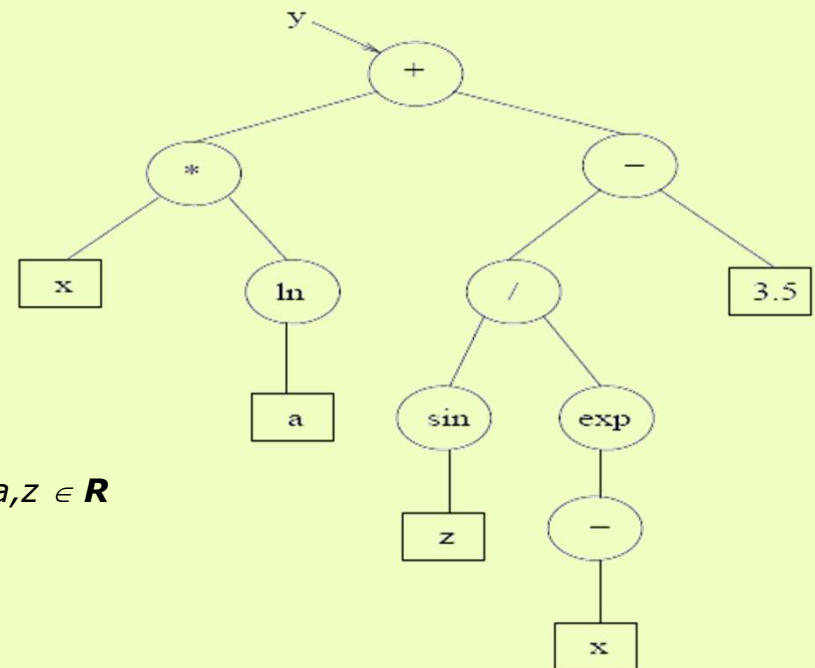


## □ Reprezentare

### ■ exemplu de evoluare a unei expresii aritmetice

- Problemă: să se determine expresia aritmetică identificată prin datele:

X	a	z	Output
1.5	2	0.7	0.52690
0.8	0.25	2	-2.48536
2	1	0.3	-1.21638



- Setul de funcții  $F = \{+, -, *, /, \sin, \exp, \ln\}$
- Setul de terminale  $T = \{x, a, z, 3.14\}$ , cu  $x, a, z \in \mathbf{R}$
- Soluție:  $y = x * \ln(a) + \sin(z) / \exp(-x) - 3.4$

# Programarea genetică – reprezentare și operatori



## □ Inițializarea cromozomilor

### ■ Aleatoare, respectând

- O limită a adâncimii maxime
- Semantica dată de gramatică

### ■ Problema “bloat” – supraviețuirea arborilor foarte mari

### ■ Metode

#### □ Metoda *Full* – arbori compleți

- Nodurile de la adâncimea  $d < D_{\max}$  se inițializează aleator cu o funcție din setul de funcții  $F$
- Nodurile de la adâncimea  $d = D_{\max}$  se inițializează aleator cu un terminal din setul de terminale  $T$

#### □ Metoda *Grow* – arbori incompleți

- Nodurile de la adâncimea  $d < D_{\max}$  se inițializează aleator cu un element din  $F \cup T$
- Nodurile de la adâncimea  $d = D_{\max}$  se inițializează aleator cu un terminal din setul de terminale  $T$

#### □ Metoda *Ramped half and half*

- $\frac{1}{2}$  din populație se creează cu metoda *Full*
- $\frac{1}{2}$  din populație se creează cu metoda *Grow*
- Folosind diferite adâncimi

# Programarea genetică – reprezentare și operatori



## □ Evaluarea cromozomilor

- Necesitatea datelor de antrenament (cazuri de testare)
- Calculul diferenței între ce trebuie obținut și ceea ce se obține de fapt
  - Expresii de tip Boolean → numărul ieșirilor corect prezise
  - Expresii aritmetice → media pătratelor diferențelor între ieșirile corecte și ieșirile prezise
  - Programe → numărul datelor de test corect procesate
- Criteriul de optim → minimizare
- Evaluarea poate penaliza:
  - Soluțiile invalide
  - Dimensiunea (prea mare a) arborilor

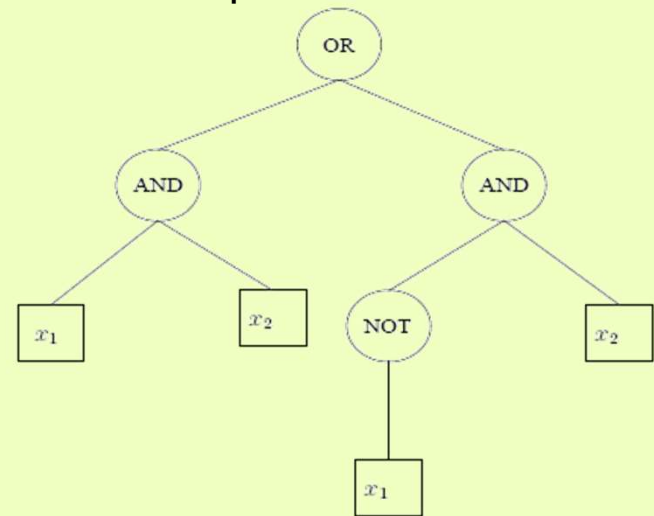
# Programarea genetică – reprezentare și operatori



## ❑ Evaluarea cromozomilor – exemplu

- Problemă: să se determine expresia logică identificată prin date:

- ❑  $C = (x_1 \text{ AND } x_2) \text{ OR } (\text{NOT } x_1 \text{ AND } x_2)$



	x1	x2	Output real	Output calculat	Eroare =  output real – output calculat
	0	0	0	0	0
	0	1	1	1	0
	1	0	1	0	1
	1	1	0	1	1
suma					2

# Programare genetică – reprezentare și operatori



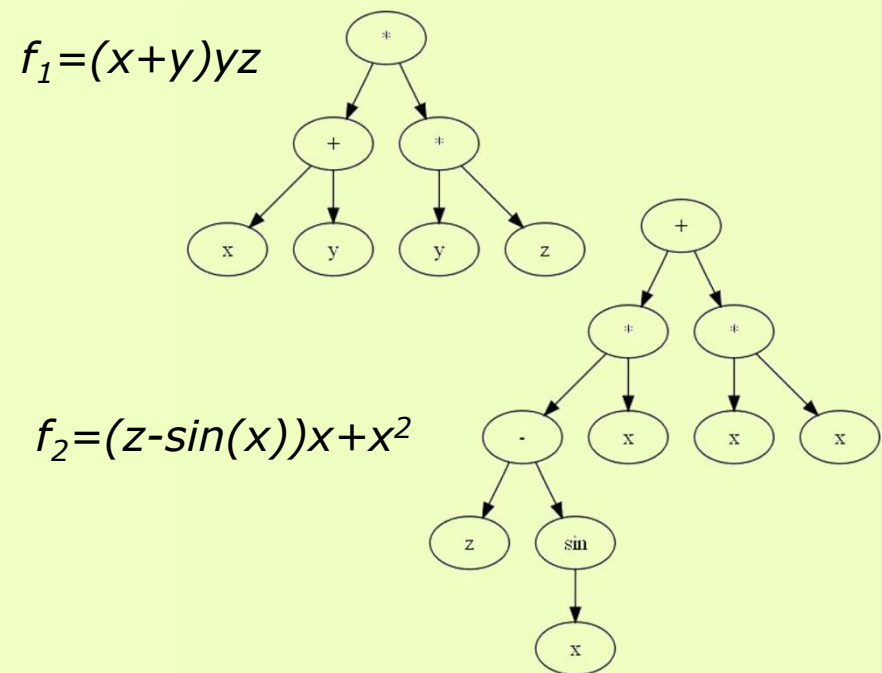
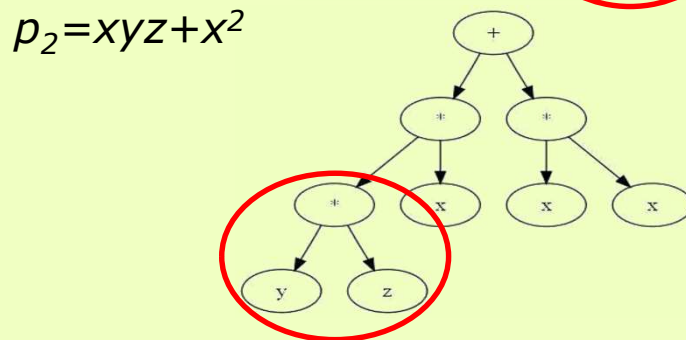
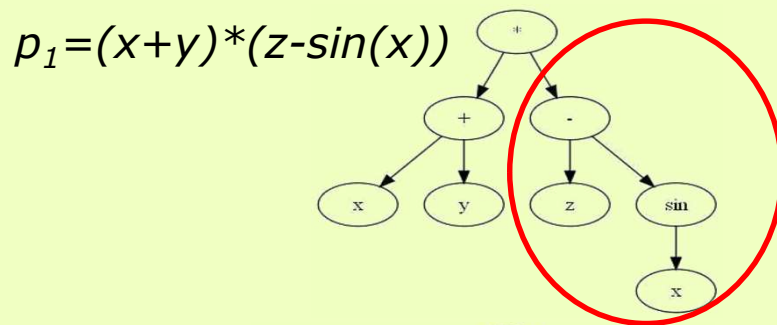
- Selecția pentru reproducere
  - Bazată pe fitness
    - Selecție proporțională (bazată pe fitness)
    - Selecție bazată pe ranguri
    - Selecție prin turnir
  - În populații foarte mari
    - Se acordă ranguri indivizilor (pe bază de fitness) și se stabilesc mai multe grupe
      - Grupa 1: cei mai buni  $x\%$  din populație
      - Grupa 2: restul de  $(100-x)\%$  din populație
    - Alegerea va fi făcută din:
      - grupa 1 în 80% din cazuri
      - grupa 2 în 20% din cazuri
      - Ex.
        - $\mu = 1000$ ,  $x = 32\%$
        - $\mu = 2000$ ,  $x = 16\%$
        - $\mu = 4000$ ,  $x = 8\%$
        - $\mu = 8000$ ,  $x = 4\%$

# Programare genetică – reprezentare și operatori



## ■ Recombinarea (încrucișarea)

- Cu punct de tăietură



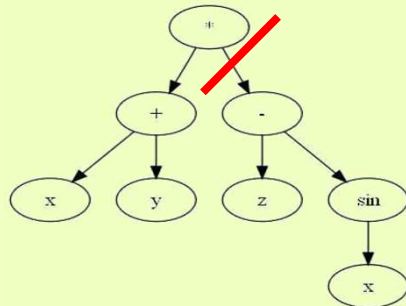
# Programare genetică – reprezentare și operatori



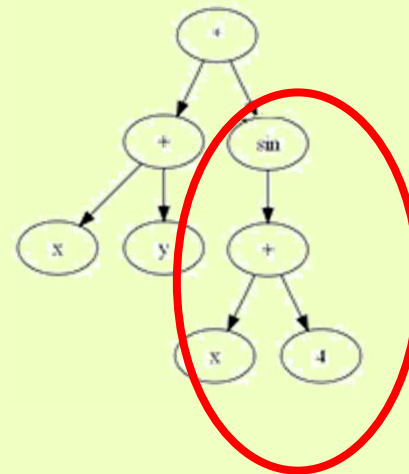
## □ Mutație

- Mutație de tip *Koza* → Înlocuirea unui nod (intern sau frunză) cu un nou sub-arbore

$$p = (x + y) * (z - \sin(x))$$



$$f = (x + y) * \sin(x + 4)$$





# Programarea genetică – proprietăți

- ❑ Folosirea cromozomilor ne-liniari
- ❑ Necesită lucrul cu populații foarte numeroase
  - Algoritmi înceți
- ❑ Comparatie AG și PG
  - Forma cromozomilor
    - ❑ AG – cromozomi liniari
    - ❑ PG – cromozomi ne-liniari
  - Dimensiunea cromozomilor
    - ❑ AG – fixă
    - ❑ PG – variabilă (în adâncime sau lățime)
  - Schema de creare a descendenților
    - ❑ AG – încrucișare și mutație
    - ❑ PG – încrucișare sau mutație



# Programare genetică – aplicații



## □ Învățare automată

- probleme de regresie
  - Predicții de curs valutar
  - Previziunea vremii
- probleme de clasificare (învățare supervizată)
  - Proiectarea circuitelor digitale
  - Recunoașterea imaginilor
  - Diagnosticare medicală
- probleme de clusterizare (învățare nesupervizată)
  - Analiza secvențelor de ADN
  - Cercetări și studii de piață (segmentarea pieței)
  - Analiza rețelelor sociale
  - Analiza rezultatelor căutărilor în Internet

# Programare genetică – variante

---



- ❑ Linear Genetic Programming
  - ❑ Gene Expression Programming
  - ❑ Cartesian Genetic Programming
  - ❑ Grammatical Evolution
  - ❑ Multi Expression Programming
  - ❑ Traceless Genetic Programming
- 
- ❑ Mai multe detalii despre GP si variantele sale în cursurile dedicate învățării automate



# Recapitulare

Generational GA	Steady – state GA
<pre> Initialization(pop) Evaluation(pop) g = 0; While (not stop_condition) do   Repeat     p1=Selection(pop)     p2=Selection(pop)     Crossover(p1,p2) =&gt; o1 and o2     Mutation(o1) =&gt; o1*     Mutation(o2) =&gt; o2*     Evaluation(o1*)     Evaluation(o2*)     Add o1* and o2* into popAux   Until popAux is full.   pop ← popAux EndWhile </pre>	<pre> Initialization(pop) Evaluation(pop)  While (not stop_condition) do    p1=Selection(pop)   p2=Selection(pop)   Crossover(p1,p2) =&gt; o1 and o2   Mutation(o1) =&gt; o1*   Mutation(o2) =&gt; o2*   Evaluation(o1*)   Evaluation(o2*)   Best(o1*,o2*) replaces Worst(pop) EndWhile </pre>
SE	PE
<pre> Initialization(pop) Evaluation(pop) g = 0; While (not stop_condition) do   Repeat     p1=Selection(pop)     p2=Selection(pop)     Crossover(p1,p2) =&gt; o1     Mutation(o1,param) =&gt; o1*, param*     Evaluation(o1*)     Add o1* into popAux   Until popAux contains λ cromozoms   pop ← Bestμ(popAux) //SE(μ,λ)   pop ← Bestμ(popUpopAux) //SE(μ+λ) EndWhile </pre>	<pre> Initialization(pop) Evaluation(pop) g = 0; While (not stop_condition) do   For all cromozoms c from pop      Mutation(c,param) =&gt; o1*, param*     Evaluation(o1*)     Add o1* into popAux    pop ← RoundRobin(popAux) EndWhile </pre>



# Recapitulare

---

- Reprezentare și fitness
  - Dependente de problemă
  
- Operatori de căutare
  - Selecția pentru reproducere și pentru supraviețuire
    - Dependentă de fitness
    - Independentă de reprezentare
  - Încrucișarea și mutația
    - Dependente de reprezentare
    - Independente de fitness
    - Probabilitatea de încrucișare
      - Acționează la nivel de cromozom
    - Probabilitatea de mutație
      - Acționează la nivel de genă

- 
- Informațiile prezentate au fost colectate din diferite surse de pe internet, precum și din cursurile de inteligență artificială ținute în anii anteriori de către:
    - Conf. Dr. Mihai Oltean – [www.cs.ubbcluj.ro/~moltean](http://www.cs.ubbcluj.ro/~moltean)
    - Lect. Dr. Crina Groșan - [www.cs.ubbcluj.ro/~cgrosan](http://www.cs.ubbcluj.ro/~cgrosan)
    - Prof. Dr. Horia F. Pop - [www.cs.ubbcluj.ro/~hfpop](http://www.cs.ubbcluj.ro/~hfpop)