



UNIVERSITATEA BABEȘ-BOLYAI
Facultatea de Matematică și Informatică



INTELIGENȚĂ ARTIFICIALĂ

Rezolvarea problemelor de căutare

Strategii de căutare informată locală

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

Sumar

- Rezolvarea problemelor prin căutare
 - Strategii de căutare informate (euristice) – SCI
 - Strategii locale
 - Algoritmi evolutivi

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)

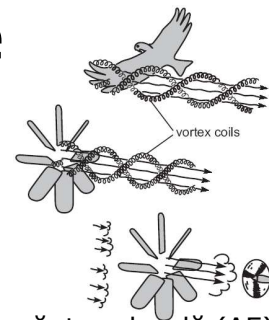
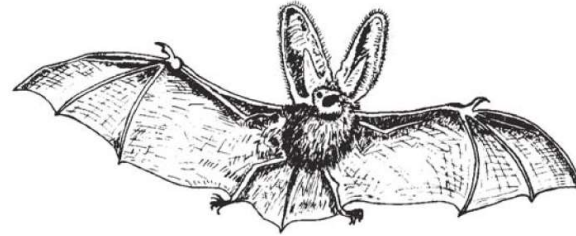
Algoritmi inspirați de natură

- ❑ Care este cea mai bună metodă de rezolvare a unei probleme?
 - Creierul uman
 - ❑ a creat roata, mașina, orașul, etc
 - Mecanismul evoluției
 - ❑ a creat creierul (mintea) umană
- ❑ Simularea naturii
 - Cu ajutorul mașinilor → rețelele neuronale artificiale simulează mintea umană
 - ❑ mașini de zbor, computere bazate pe ADN, computere cu membrane
 - Cu ajutorul algoritmilor
 - ❑ algoritmi evolutivi simulează evoluția naturii
 - ❑ algoritmi inspirați de comportamentul de grup simulează adaptarea colectivă și procesele sociale dintr-un colectiv (*Particle Swarm Optimisation*)
 - ❑ algoritmi inspirați de furnici (*Ant Colony Optimisation*)

Algoritmi evolutivi – aspecte teoretice

□ Simularea naturii

- Zborul liliecilor
- Leonardo da Vinci – schema unei mașini de zbor
- Zborul păsărilor și al avioanelor
- Zborul păsărilor și turbinele eoliene



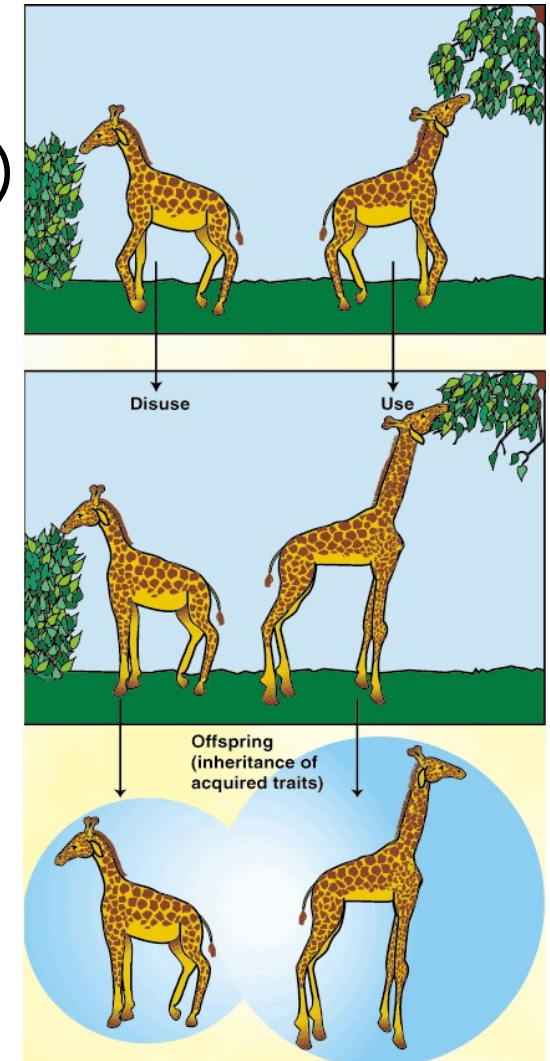
Algoritmi evolutivi – aspecte teoretice

- ❑ Care sunt caracteristicile de bază ale AE?
 - Implică procese iterative și paralele
 - Folosesc populații de potențiale soluții
 - Se bazează pe o căutare aleatoare
 - Sunt inspirați de biologie – implică mecanisme precum:
 - ❑ selecția naturală
 - ❑ reproducerea
 - ❑ recombinarea
 - ❑ mutația

Algoritmi evolutivi – aspecte teoretice

Câteva repere istorice

- Jean Baptise de Lamarck (1744-1829)
 - A propus în 1809 o explicație pentru originea speciilor în cartea *Zoological Philosophy*:
 - Nevoile unui organism determină caracteristicile care evoluează
 - Caracteristicile utile dobândite în cursul vieții unui organism se pot transfera urmașilor acestuia
 - Legea utilizării și neutilizării
 - *use and disuse*

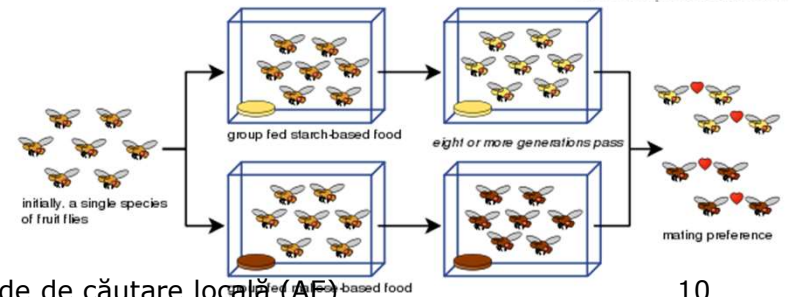
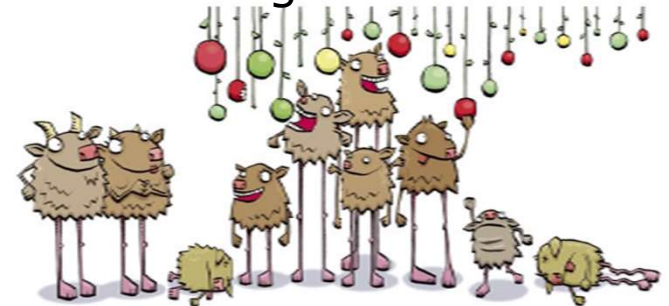


Algoritmi evolutivi – aspecte teoretice

Câteva repere istorice

- Charles Darwin (1807-1882)
 - În cartea *Origin of Species* demonstrează că toate organismele au evoluat din alte organisme pe baza:

- variației
 - supraproducția de descendenți
- selecției naturale
 - competiția (generații constante ca dimensiune)
 - supraviețuirea pe baza calității/adaptării la mediul de viață (fitness)
 - reproducerea
 - apariția de specii noi



Algoritmi evolutivi – aspecte teoretice

Câteva repere istorice

■ Teoria evolutivă modernă

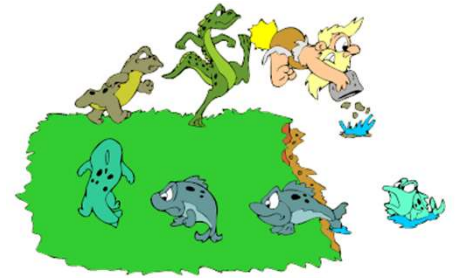
- Îmbogățește teoria Darwiniană cu mecanismul moștenirii genetice
- Variația genetică se produce prin:
 - mutație – spontană – și
 - reproducere sexuală
- L. Fogel 1962 (San Diego, CA) → programare evolutivă – PE – (*Evolutionary Programming*)
- J. Holland 1962 (Ann Arbor, MI) → algoritmi genetici – AG – (*Genetic Algorithms*)
- I. Rechenberg & H.-P. Schwefel 1965 (Berlin, Germany) → strategii evolutive – SE – (*Evolution Strategies*)
- J. Koza 1989 (Palo Alto, CA) → programare genetică – PG – (*Genetic Programming*)

Algoritmi evolutivi – aspecte teoretice

□ Metafora evolutivă

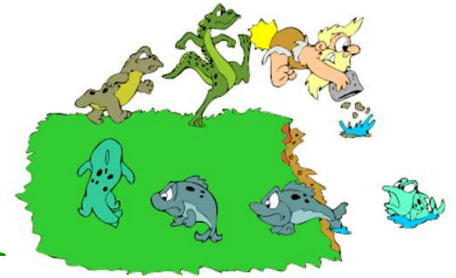
Evoluția naturală		Rezolvarea problemelor
Individ	↔	Soluție potențială
Populație	↔	Mulțime de soluții potențiale
Cromozom	↔	Codarea unei soluții potențiale
Genă	↔	Parte a codării
Fitness	↔	Calitate
Încrucișare și mutație	↔	Operatori de căutare
Mediu	↔	Problemă

Algoritmi evolutivi - algoritm



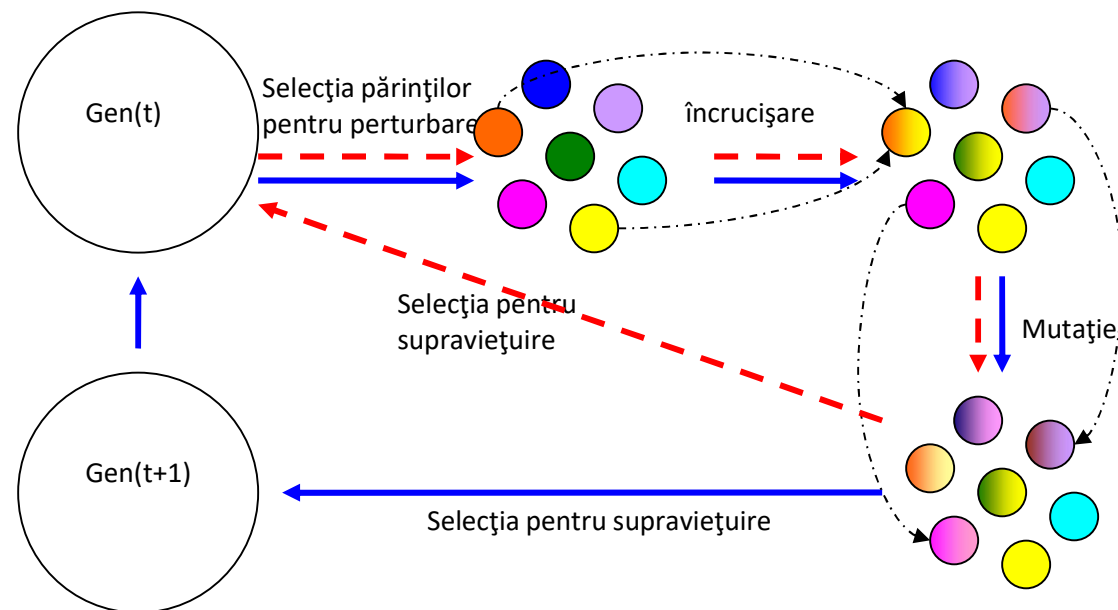
- Schema generală
- Proiectare

Algoritmi evolutivi – algoritmi

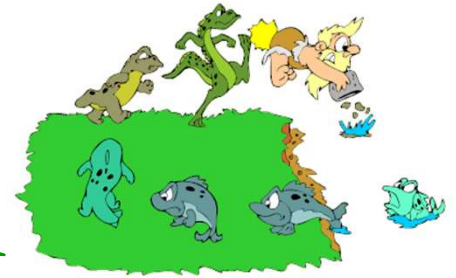


□ Schema generală a unui AE

- Generațional 
- Steady-state 



Algoritmi evolutivi – algoritmi



□ Proiectare

- Alegerea unei reprezentări a cromozomilor
- Alegerea unui model de populație
- Stabilirea unei funcții de evaluare
- Stabilirea operatorilor genetici
 - Selecție
 - Mutație
 - Recombinare
- Stabilirea unui criteriu de stop

Algoritmi evolutivi – algoritm

Proiectare – alegerea unei reprezentări

❑ 2 nivele de existență pentru o soluție candidat

■ Nivel exterior → fenotip

- ❑ Individ - obiectul original în contextul dat de problemă
- ❑ Aici are loc evaluarea unei potențiale soluții
- ❑ Furnică, rucsac, elefant, orașe, ...



■ Nivel interior → genotip

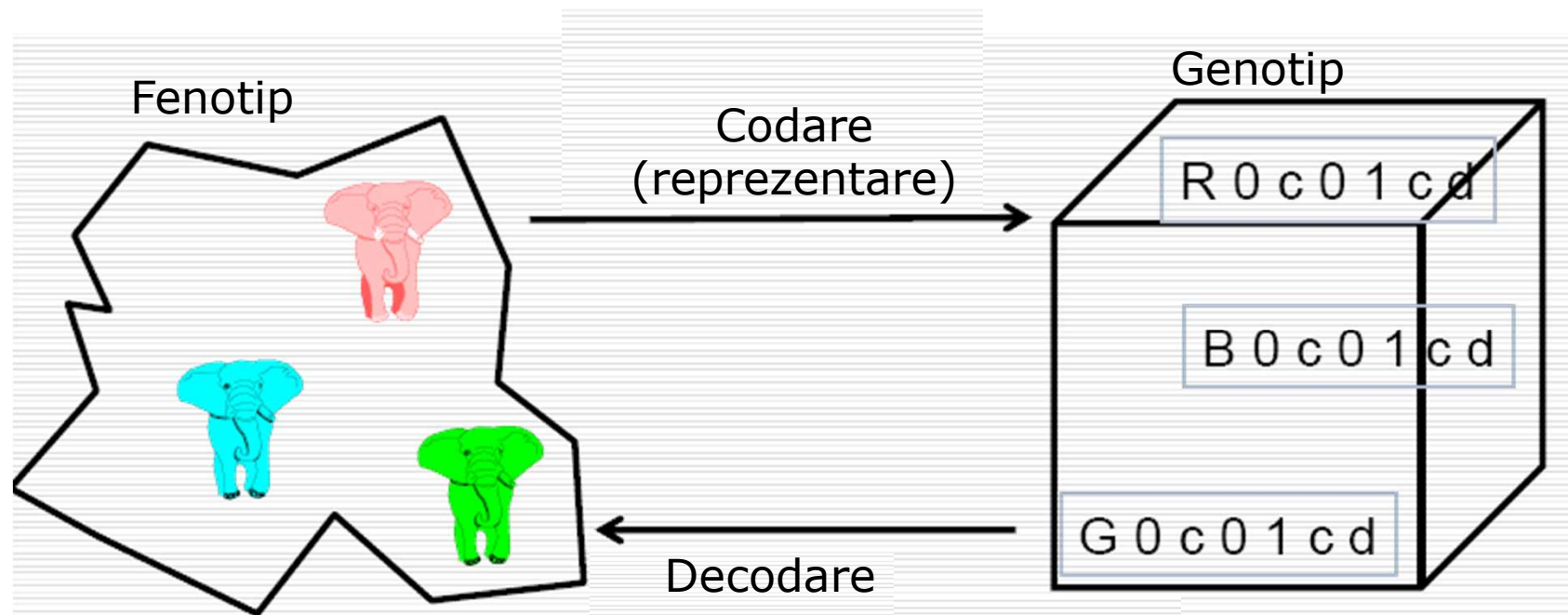
- ❑ Cromozom – codul asociat unui obiect
 - format din gene, poziționate în locuri (fixe) – loci – și având anumite valori – alele
- ❑ Aici are loc căutarea unei noi potențiale soluții
- ❑ Vector unidimensional (numeric, boolean, string), matrice, ...



Algoritmi evolutivi – algoritm

Proiectare – alegerea unei reprezentări

- ❑ Reprezentarea trebuie să fie relevantă pentru:
 - problemă,
 - funcția de evaluare și
 - operatorii genetici



Algoritmi evolutivi – algoritm

Proiectare – alegerea unei reprezentări

Tipologia reprezentării cromozomilor

□ Liniară

■ Discretă

- Binară → problema rucsacului

- Ne-binară

 - Întreagă

 - Oarecare → procesarea imaginilor

 - Permutări → problema comisului voiajor

 - Categoricală → problema colorării hărților

- Continuă (reală) → optimizări de funcții

□ Arborescentă → probleme de regresie

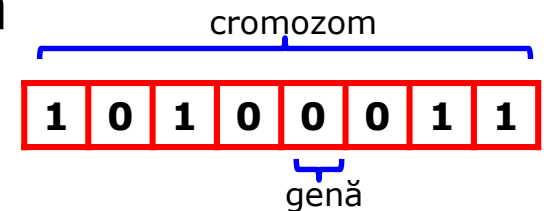
Algoritmi evolutivi – algoritm

Proiectare – alegerea unei reprezentări

□ Reprezentare liniară discretă binară

■ Genotip

□ șir de biți



Algoritmi evolutivi – algoritm

Proiectare – alegerea unei reprezentări

□ Reprezentare liniară discretă binară

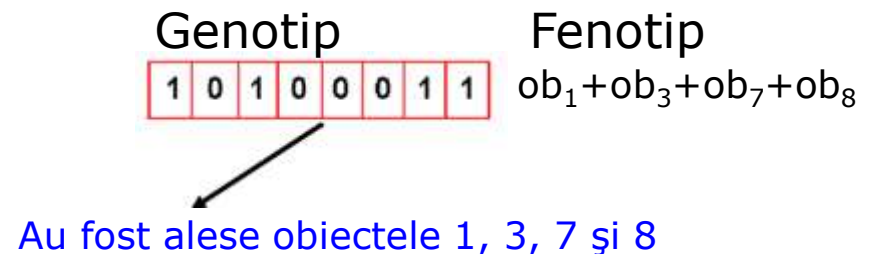
■ Genotip

- șir de biți

■ Fenotip

- Elemente de tip Boolean

- Ex. Problema rucsacului – obiectele alese pentru umplerea rucsacului



Algoritmi evolutivi – algoritm

Proiectare – alegerea unei reprezentări

□ Reprezentare liniară discretă binară

■ Genotip

- șir de biți

■ Fenotip

- Elemente de tip Boolean

- Ex. Problema rucsacului – obiectele alese pentru umplerea rucsacului

- Numere întregi

Genotip Fenotip

1	0	1	0	0	0	1	1
---	---	---	---	---	---	---	---

= 163

$1 \cdot 2^7 + 0 \cdot 2^6 + 1 \cdot 2^5 + 0 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 =$
 $128 + 32 + 2 + 1 = 163$

Algoritmi evolutivi – algoritm

Proiectare – alegerea unei reprezentări

□ Reprezentare liniară discretă binară

■ Genotip

- șir de biți

■ Fenotip

- Elemente de tip Boolean

- Ex. Problema rucsacului – obiectele alese pentru umplerea rucsacului

- Numere întregi

- Numere reale într-un anumit Interval (ex. [2.5, 20.5])

Genotip

1 0 1 0 0 0 1 1

Fenotip

= 13.9609

$$x = 2.5 + \frac{163}{256} (20.5 - 2.5) = 13.9609$$

Algoritmi evolutivi – algoritm

Proiectare – alegerea unei reprezentări

Transformarea valorilor reale reprezentate pe biți

□ Fie $z \in [x, y] \subseteq \mathcal{R}$ reprezentat ca $\{a_1, \dots, a_L\} \in \{0, 1\}^L$

□ Funcția $[x, y] \rightarrow \{0, 1\}^L$ trebuie să fie inversabilă (un fenotip corespunde unui genotip)

□ Funcția $\Gamma: \{0, 1\}^L \rightarrow [x, y]$ definește reprezentarea

$$\Gamma(a_1, \dots, a_L) = x + \frac{y - x}{2^L - 1} \cdot \left(\sum_{j=0}^{L-1} a_{L-j} \cdot 2^j \right) \in [x, y]$$

□ Observații

- Se pot reprezenta doar 2^L valori
- L indică precizia maximă a soluției
- Pentru o precizie cât mai bună → cromozomi lungi → evoluție încetinită

Algoritmi evolutivi – algoritm

Proiectare – alegerea unei reprezentări

- Reprezentare liniară discretă ne-binară întreagă oarecare
 - Genotip
 - șir de numere întregi dintr-un anumit interval
 - Fenotip
 - Utilitatea numerelor în problemă

- Ex. Problema plății unei sume folosind diferite monezi
 - Genotip → șir de nr întregi de lungime egală cu numărul de monezi diferite, fiecare număr din intervalul $[0, \text{suma}/\text{valoarea monezii curente}]$
 - Fenotip → câte monezi din fiecare tip trebuie considerate

Algoritmi evolutivi – algoritm

Proiectare – alegerea unei reprezentări

- Reprezentare liniară discretă ne-binară întreagă de tip permutare
 - Genotip
 - Permutare de n numere (n – numărul de gene)
 - Fenotip
 - Utilitatea permutării în problemă
- Ex. Problema comisului voiajor
 - Genotip → permutare de n elemente
 - Fenotip → ordinea de vizitare a orașelor, știind că fiecărui oraș îi corespunde un număr din mulțimea $\{1, 2, \dots, n\}$

Algoritmi evolutivi – algoritm

Proiectare – alegerea unei reprezentări

- ❑ Reprezentare liniară discretă ne-binară categorială
 - Similară cu cea întreagă, dar în loc de numere se folosesc etichete
 - Genotip
 - ❑ șir de etichete dintr-o anumită mulțime
 - Fenotip
 - ❑ Interpretarea etichetelor
- Ex. Problema colorării hărților
 - ❑ Genotip → șir de etichete (culori) de lungime egală cu numărul de țări, fiecare etichetă aparținând unei mulțimi de culori date
 - ❑ Fenotip → cu ce culoare trebuie hașurată fiecare hartă a unei țări

Algoritmi evolutivi – algoritm

Proiectare – alegerea unei reprezentări

- Reprezentare liniară continuă (reală)
 - Genotip
 - Șir de numere reale
 - Fenotip
 - Utilitatea numerelor în problemă

- Ex. Problema optimizării funcțiilor $f: R^n \rightarrow R$
 - Genotip \rightarrow tuplu de numere reale $X=[x_1, x_2, \dots, x_n]$, $x_i \in R$
 - Fenotip \rightarrow valorile asociate argumentelor funcției f

Algoritmi evolutivi – algoritm

Proiectare – alegerea unei reprezentări

□ Reprezentare arborescentă

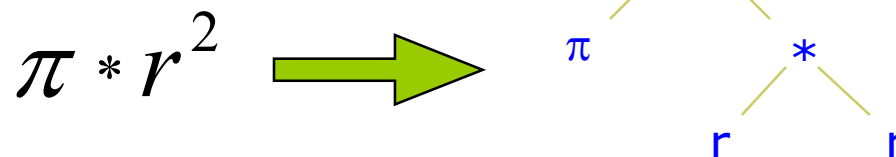
■ Genotip

- Arbori care codează S-Expresii
- Nodurile interne ale arborelui → funcții (F)
 - Matematici
 - Operatori aritmetici
 - Operatori de tip Boolean
 - Instrucțiuni
 - Într-un limbaj de programare
 - Alt tip de instrucțiuni
- Frunzele arborelui → terminale (T)
 - Valori reale sau Booleene, constante sau variabile
 - Subprograme

■ Fenotip

- Interpretarea S-expresiilor

■ Ex. Calculul ariei unui cerc



Algoritmi evolutivi – algoritm

Proiectare – formarea unei populații

□ Populație – concept

■ Scop

- reține o colecție de soluții candidat
 - se permit repetiții
- este folosită în întregime în procesul de selecție pentru reproducere

■ Proprietăți

- dimensiune (de obicei) fixă μ
- diversitate
 - Nr de fitness-uri/fenotipuri/genotipuri diferite

■ Observații

- Reprezintă unitatea de bază care evoluează
 - populația întreagă evoluează, nu indivizii!!!

Algoritmi evolutivi – algoritm

Proiectare – formarea unei populații

- Populație – inițializare

- Uniformă (dacă e posibil) în spațiul de căutare

- Stringuri binare

- generarea de 0 și 1 cu probabilitatea 0.5

- Șiruri de numere reale generate uniform (într-un anumit interval)

- Permutări

- generarea permutării identice și efectuarea unor schimbări

Algoritmi evolutivi – algoritm

Proiectare – formarea unei populații

□ Populație – inițializare

- Uniformă (dacă e posibil) în spațiul de căutare
 - Arbori
 - 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

Algoritmi evolutivi – algoritm

Proiectare – formarea unei populații

- Modele de populații – algoritm evolutiv:
 - Generațional
 - În fiecare generație se crează μ descendenți
 - Fiecare individ supraviețuiește o singură generație
 - Mulțimea părinților este înlocuită în întregime cu mulțimea descendenților
 - Steady-state
 - În fiecare generație se obține un singur descendent
 - Un singur părinte (cel mai slab) este înlocuit cu descendentul obținut
- Discrepanța între generații (*Generation Gap*)
 - Proporția populației înlocuite
 - $1 = \mu/\mu$, pentru modelul generațional
 - $1/\mu$, pentru modelul steady-state

Algoritmi evolutivi – algoritm

Proiectare – funcția de evaluare

□ Scop

- Reflectă condițiile la care trebuie să se adapteze populația
- Funcție de calitate sau funcție obiectiv
- Asociază o valoare fiecărei soluții candidat
 - Consecințe asupra selecției → cu cât sunt mai multe valori diferite, cu atât e mai bine

□ Proprietăți

- Etapa cea mai costisitoare
 - Nu se re-evaluează indivizii nemodificați

□ Tipologie:

- După nr de obiective urmărite:
 - Uni-obiectiv
 - Multi-obiectiv → fronturi Pareto
- După direcția optimizării
 - De maximizat
 - De minimizat
- După gradul de exactitate
 - Exactă
 - Euristică

Algoritmi evolutivi – algoritm

Proiectare – funcția de evaluare

□ Exemple

■ Problema rucsacului

- reprezentare → liniară discretă binară
- fitness → $\text{abs}(\text{greutatea rucsacului} - \text{greutatea obiectelor alese}) \rightarrow$ minimizare

■ Problema plății unei sume folosind diferite monezi

- reprezentare → liniară discretă întreagă
- fitness → $\text{abs}(\text{suma de plată} - \text{suma monezilor selectate}) \rightarrow$ minimizare

■ Problema comisului voiaior

- reprezentare → liniară discretă întreagă sub formă de permutare
- fitness → costul drumului parcurs → minimizare

■ Problema optimizării funcțiilor

- Reprezentare → liniară continuă reală
- fitness → valoarea funcției → minimizare/maximizare

■ Calculul ariei unui cerc

- reprezentare → arborescentă
- fitness → suma pătratelor erorilor (diferențelor între valoarea reală și cea calculată pe un set de exemple) → minimizare

Algoritmi evolutivi – algoritm

Proiectare – selecția



- Scop:
 - acordă șanse de reproducere/supraviețuire mai mari indivizilor mai buni
 - și indivizii mai slabi trebuie să aibă șansa să se reproducă/supraviețuiască pentru că pot conține material genetic util
 - direcționează populația spre îmbunătățirea calității
- Proprietăți
 - lucrează la nivel de populație
 - se bazează doar pe fitnessul indivizilor (este independentă de reprezentare)
 - ajută la evadarea din optimele locale datorită naturii sale stocastice

Algoritmi evolutivi – algoritm

Proiectare – selecția



□ Tipologie

■ În funcție de scop:

- Selecția părinților (din generația curentă) pentru reproducere
- Selecția supraviețuitorilor (din părinți și descendenți) pentru generația următoare

■ În funcție de modul de decidere al câștigătorului

- Deterministă – cel mai bun câștigă
- Stocastică – cel mai bun are cele mai mari șanse să câștige

■ În funcție de mecanism

- Selecția pentru reproducere
 - Selecție proporțională (bazată pe fitness)
 - Selecție bazată pe ranguri
 - Selecție prin turnir -----> Bazată pe o parte din populație
- Selecția pentru supraviețuire
 - Bazată pe vârstă
 - Bazată pe calitate (fitness)

Algoritmi evolutivi – algoritm

Proiectare – selecția pt. reproducere



□ Selecție proporțională (bazată pe fitness) – SP

□ Ideea de bază

- Algoritmul ruletei la nivelul întregii populații
- Estimarea numărului de copii ale unui individ

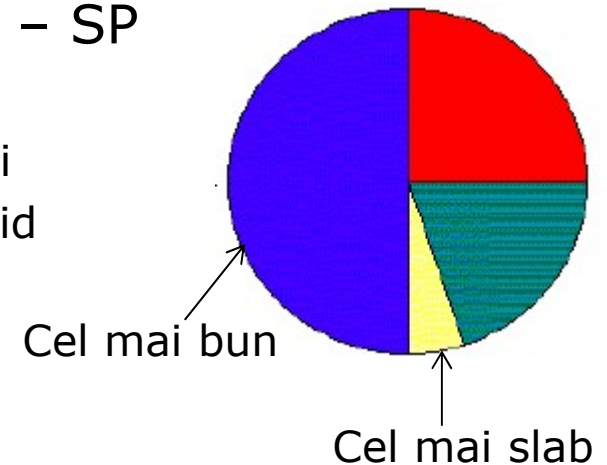
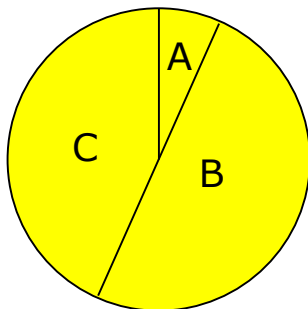
$$E(n_i) = \mu \frac{f(i)}{\langle f \rangle}, \text{ unde:}$$

- μ = dimensiunea populației,
- $f(i)$ = fitnessul individului i ,
- $\langle f \rangle$ = fitnessul mediu al populației

□ Indivizii mai buni

- au alocat mai mult spațiu în ruletă
- au șanse mai mari să fie selectați

□ Ex. O populație cu $\mu = 3$ indivizi



	$f(i)$	$P_{\text{selSP}}(i)$
A	1	$1/10=0.1$
B	5	$5/10=0.5$
C	4	$4/10=0.4$
Suma	10	1

Algoritmi evolutivi – algoritm

Proiectare – selecția pt. reproducere



Selecție proporțională (bazată pe fitness)

Avantaie

- Algoritm simplu

Dezavantaie

- Convergența prematură
 - cromozomii foarte buni tind să domine populația
- Presiune de selecție foarte mică atunci când fitnessurile indivizilor sunt foarte apropiate (la sfârșitul rulării)
- Susceptibilă de traspoziția funcției
- Rezultatele reale ale unei astfel de selecții diferă de distribuția probabilistică teoretică
- Lucrează cu întreaga populație

Soluții

scalarea fitnessului

Windowing

- $f'(i) = f(i) - \beta^t$, unde β este un parametru care depinde de istoria recentă a evoluției
 - ex. β este fitnessul celui mai slab individ din populația curentă (a t -a generație)

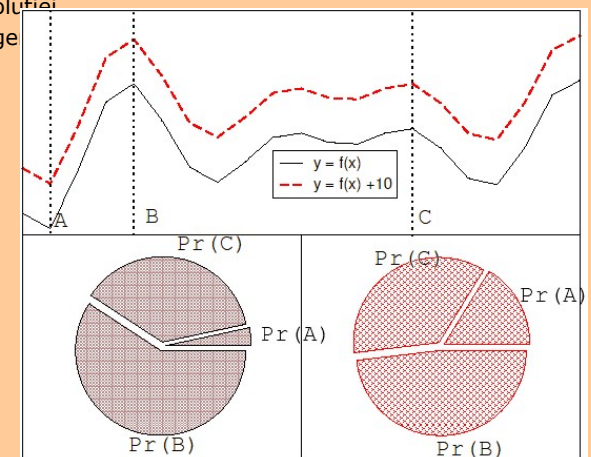
Scalare de tip sigma (de tip Goldberg)

- $f'(i) = \max\{f(i) - (\langle f \rangle - c * \sigma_f), 0.0\}$, unde:
 - c este o constantă (de obicei 2)
 - $\langle f \rangle$ - fitnessul mediu al populației
 - σ_f - deviația standard a fitnessului populației

Scalare prin normalizare

- Se începe cu fitnessurile absolute (inițiale)
- Se standardizează astfel încât Se aiustează fitnessurile a.î.:
 - ele să aparțină $[0,1]$
 - cel mai bun fitness să fie cel mai mic (egal cu 0)
 - suma lor să fie 1

alt mecanism de selecție



Algoritmi evolutivi – algoritm

Proiectare – selecția pt. reproducere



□ Selecția bazată pe ranguri – SR

■ Ideea de bază

- Se ordonează întreaga populație pe baza fitnessului
 - Crește puțin complexitatea algoritmului, dar se poate neglija această creștere comparativ cu timpul necesar evaluării unui individ
- Se acordă ranguri fiecărui individ
- Se calculează probabilitățile de selecție pe baza rangurilor
 - Cel mai slab individ are rangul 1
 - Cel mai bun individ are rangul μ
- Încearcă să rezolve problemele selecției proporționale prin folosirea fitnessurilor relative (în locul celor absolute)

Algoritmi evolutivi – algoritm

Proiectare – selecția pt. reproducere



□ Selecția bazată pe ranguri – SR

■ Modalități de acordare a rangurilor

□ Liniară (RL)
$$P_{lin_rank}(i) = \frac{2-s}{\mu} + \frac{2i(s-1)}{\mu(\mu-1)}$$

- s – presiunea de selecție
 - măsoară avantajele celui mai bun individ
 - $1.0 < s \leq 2.0$
 - în algoritmul genetic generațional s este numărul de copii ai unui individ
- Ex. pentru o populație cu $\mu = 3$ indivizi

	$f(i)$	$P_{selSP}(i)$	Rang	$P_{selRL}(i)$ pt. $s=2$	$P_{selRL}(i)$ pt. $s=1$
A	1	$1/10=0.1$	1	0.33	0.33
B	5	$5/10=0.5$	3	1.00	0.33
C	4	$4/10=0.4$	2	0.67	0.33
Suma	10	1			

□ Exponențială (RE)
$$P_{exp_rank}(i) = \frac{1-e^{-i}}{c}$$

- Cel mai bun individ poate avea mai mult de 2 copii
- c – factor de normalizare
 - depinde de dimensiunea populației (μ)
 - trebuie ales a.î. suma probabilităților de selecție să fie 1

Algoritmi evolutivi – algoritm

Proiectare – selecția pt. reproducere



- Selecția bazată pe ranguri – SR
 - Avantaje
 - Păstrează presiunea de selecție constantă
 - Dezavantaje
 - Lucrează cu întreaga populație
 - Soluții
 - Alt mecanism de selecție

Algoritmi evolutivi – algoritm

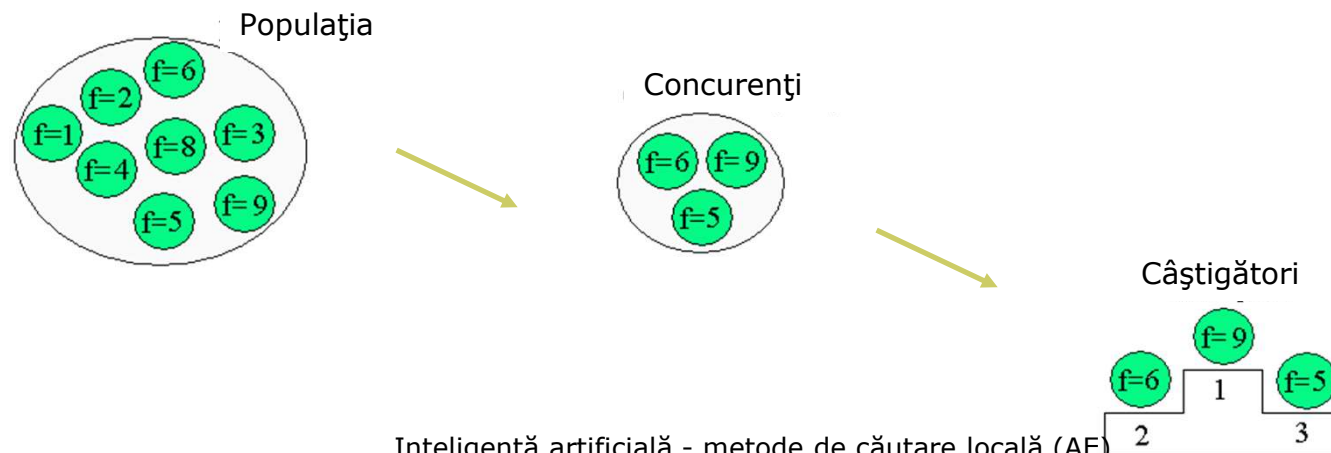
Proiectare – selecția pt. reproducere



□ Selecția prin turnir

■ Ideea de bază

- Se aleg aleator k indivizi \rightarrow eșantion de k indivizi (k – mărimea turnirului)
- Se selectează cel mai bun individ dintre cei aleși anterior
- Probabilitatea alegerii unui individ în eșantion depinde de
 - Rangul individului
 - Dimensiunea eșantionului (k)
 - Cu cât k este mai mare, cu atât crește și presiunea de selecție
 - Modul în care se face alegerea – dacă se realizează cu înlocuire (model steady-state) sau nu
 - Alegerea fără înlocuire crește presiunea de selecție
 - Pt $k = 2$ timpul necesar ca cel mai bun individ să domine populația este același cu cel de la selecția pe bază de ranguri liniare cu $s = 2 * p$, p – probabilitatea alegerii celui mai bun individ din populație



Algoritmi evolutivi – algoritm

Proiectare – selecția pt. reproducere



□ Selecția prin turnir

■ Avantaje

- Nu implică lucrul cu întreaga populație
- Ușor de implementat
- Ușor de controlat presiunea de selecție prin intermediul parametrului k

■ Dezavantaje

- Rezultatele reale ale unei astfel de selecții diferă de distribuția probabilistică teoretică (similar selecției prin mecanismul ruletei)

Algoritmi evolutivi

Proiectare – selecția



- ❑ Selecția pentru supraviețuire (înlocuire)
 - Pe baza vârstei
 - ❑ eliminarea celor mai “bătrâni” indivizi
 - Pe baza calității (fitness-ului)
 - ❑ selecției proporțională
 - ❑ selecție bazată pe ranguri
 - ❑ selecție prin turnir
 - ❑ elitism
 - Păstrarea celor mai buni indivizi de la o generație la alta (dacă descendenții sunt mai slabi ca părinții se păstrează părinții)
 - ❑ GENITOR (înlocuirea celui mai slab individ)
 - Eliminarea celor mai slabi λ indivizi

Algoritmi evolutivi - algoritm

Proiectare – operatori de variație



- Scop:
 - Generarea unor soluții potențiale noi

- Proprietăți
 - lucrează la nivel de individ
 - se bazează doar pe reprezentarea indivizilor (independent de fitness)
 - Ajută la explorarea și exploatarea spațiului de căutare
 - Trebuie să producă indivizi valizi

- Tipologie
 - În funcție de aritate
 - Aritate 1 → operatori de mutație
 - Aritate > 1 → operatori de recombinare/încrucișare

Algoritmi evolutivi – algoritm

Proiectare – mutația



□ Scop

- Reintroducerea în populație a materialului genetic pierdut
- Operator unar de căutare (spațiul continuu)
- Introducerea diversității în populație (în spațiul discret – binar)

□ Proprietăți

- Acționează la nivel de genotip
- Bazată pe elemente aleatoare

înainte

1 1 1 1 1 1

după

1 1 1 0 1 1



- Responsabilă cu explorarea unor noi regiuni promițătoare ale spațiului de căutare
- Este responsabilă de evadarea din optimele locale
- Trebuie să producă mici schimbări stocastice ale individului
- Mărimea mutației trebuie să fie controlabilă
- Se produce cu o anumită probabilitate (p_m) la nivelul fiecărei gene a unui cromozom

Algoritmi evolutivi – algoritm

Proiectare – mutația



□ Tipologie

- Reprezentare binară
 - Mutație tare - bit-flipping
 - Mutație slabă
- Reprezentare întreagă
 - Random resetting
 - Creep mutation
- Reprezentare permutare
 - Mutație prin inserție
 - Mutație prin interchimbare
 - Mutație prin inversare
 - Mutație prin amestec
 - Mutație k-opt
- Reprezentare reală
 - Mutație uniformă
 - Mutație neuniformă
 - Mutație Gaussiană
 - Mutație Cauchy
 - Mutație Laplace
- Reprezentare arborescentă → într-un curs viitor
 - Mutație grow
 - mutație shrink
 - Mutație switch
 - Mutație cycle
 - Mutație tip Koza
 - Mutație pentru terminalele numerice

Algoritmi evolutivi – algoritm

Proiectare – mutația (reprez. binară)



- Un cromozom $c=(g_1, g_2, \dots, g_L)$ devine $c'=(g_1', g_2', \dots, g_L')$, unde $g_i, g_i' \in \{0, 1\}$, pt. $i=1, 2, \dots, L$
- Mutație tare – *bit flipping*
 - Ideea de bază
 - Schimbarea cu probabilitatea p_m (rată de mutație) a unor gene în complementul lor
 - $1 \rightarrow 0$
 - $0 \rightarrow 1$
 - Ex. Un cromozom cu $L = 8$ gene, $p_m = 0.1$



Algoritmi evolutivi – algoritm

Proiectare – mutația (reprez. binară)



- Un cromozom $c=(g_1, g_2, \dots, g_L)$ devine $c'=(g_1', g_2', \dots, g_L')$, unde $g_i, g_i' \in \{0, 1\}$, pt. $i=1, 2, \dots, L$

- Mutație slabă
 - Ideea de bază
 - Schimbarea cu probabilitatea p_m (rată de mutație) a unor gene în 0 sau 1
 - $1 \rightarrow 0/1$
 - $0 \rightarrow 1/0$
 - Ex. Un cromozom cu $L = 8$ gene, $p_m = 0.1$



Algoritmi evolutivi – algoritm

Proiectare – mutația (reprez. întreagă)



- Un cromozom $c=(g_1, g_2, \dots, g_L)$ devine $c'=(g_1', g_2', \dots, g_L')$, unde $g_i, g_i' \in \{val_1, val_2, \dots, val_k\}$, pt. $i=1, 2, \dots, L$
- Mutație *random resetting*
 - Ideea de bază
 - Valoarea unei gene este schimbată (cu probabilitatea p_m) într-o altă valoare (din setul de valori posibile)

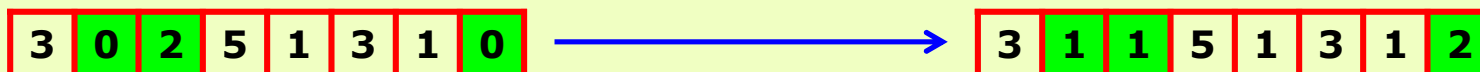


Algoritmi evolutivi – algoritm

Proiectare – mutația (reprez. întreagă)



- Un cromozom $c=(g_1, g_2, \dots, g_L)$ devine $c'=(g_1', g_2', \dots, g_L')$, unde $g_i, g_i' \in \{val_1, val_2, \dots, val_k\}$, pt. $i=1, 2, \dots, L$
- Mutație *creep*
 - Ideea de bază
 - Valoarea unei gene este schimbată (cu probabilitatea p_m) prin adăugarea unei valori (pozitivă sau negativă)
 - valoarea \rightarrow face parte dintr-o distribuție simetrică față de zero
 - modificarea produsă este fină (mică)



Algoritmi evolutivi – algoritm

Proiectare – mutația (reprez. permutare)



- Un cromozom $c=(g_1, g_2, \dots, g_L)$ cu $g_i \neq g_j$ pentru orice $i \neq j$ devine $c'=(g_1', g_2', \dots, g_L')$, unde $g_i, g_i' \in \{val_1, val_2, \dots, val_L\}$, pt. $i=1, 2, \dots, L$ a.î. $g_i' \neq g_j'$ pentru orice $i \neq j$.
- Mutație prin interschimbare (*swap mutation*)
 - Ideea de bază
 - Se aleg aleator 2 gene și se interschimbă valorile lor



Algoritmi evolutivi – algoritm

Proiectare – mutația (reprez. permutare)

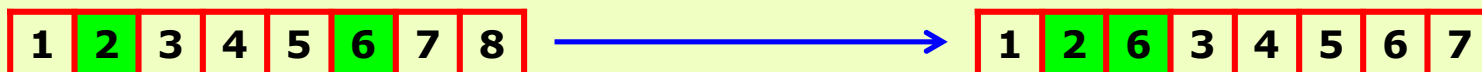


- Un cromozom $c=(g_1, g_2, \dots, g_L)$ cu $g_i \neq g_j$ pentru orice $i \neq j$ devine $c'=(g_1', g_2', \dots, g_L')$, unde $g_i, g_i' \in \{val_1, val_2, \dots, val_L\}$, pt. $i=1, 2, \dots, L$ a.î. $g_i' \neq g_j'$ pentru orice $i \neq j$.

□ Mutație prin inserție

■ Ideea de bază

- Se aleg 2 gene oarecare g_i și g_j cu $j > i$
- Se inserează g_j după g_i a.î. $g_i' = g_i$, $g_{i+1}' = g_j$, $g_{k+2}' = g_{k+1}$, pentru $k=i, i+1, i+2, \dots$



Algoritmi evolutivi – algoritm

Proiectare – mutația (reprez. permutare)

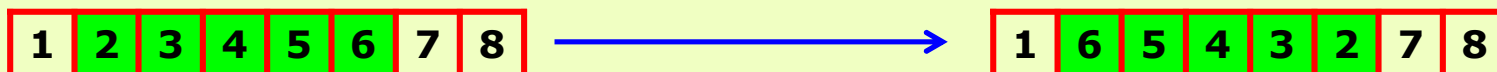


- Un cromozom $c=(g_1, g_2, \dots, g_L)$ cu $g_i \neq g_j$ pentru orice $i \neq j$ devine $c'=(g_1', g_2', \dots, g_L')$, unde $g_i, g_i' \in \{val_1, val_2, \dots, val_L\}$, pt. $i=1, 2, \dots, L$ a.î. $g_i' \neq g_j'$ pentru orice $i \neq j$.

□ Mutație prin inversare

■ Ideea de bază

- Se aleg aleator 2 gene și se inversează ordinea genelor situate între ele (substringul dintre gene)

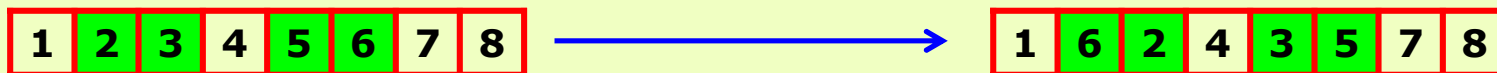


Algoritmi evolutivi – algoritm

Proiectare – mutația (reprez. permutare)



- Un cromozom $c=(g_1, g_2, \dots, g_L)$ cu $g_i \neq g_j$ pentru orice $i \neq j$ devine $c'=(g_1', g_2', \dots, g_L')$, unde $g_i, g_i' \in \{val_1, val_2, \dots, val_L\}$, pt. $i=1, 2, \dots, L$ a.î. $g_i' \neq g_j'$ pentru orice $i \neq j$.
- Mutație prin amestec (*scramble mutation*)
 - Ideea de bază
 - Se alege aleator un subșir (continuu sau discontinuu) de gene și se rearanjează acele gene



Algoritmi evolutivi – algoritm

Proiectare – mutația (reprez. permutare)

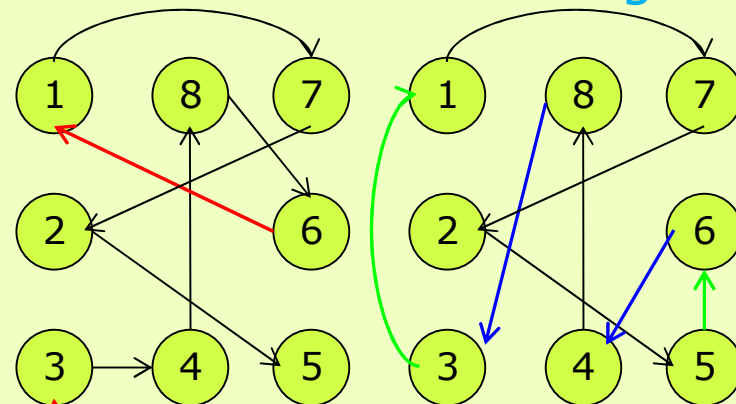
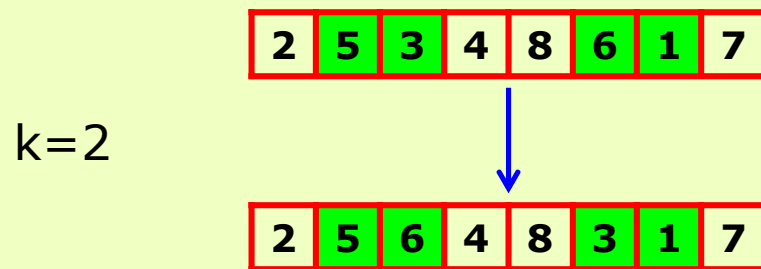


- Un cromozom $c=(g_1, g_2, \dots, g_L)$ cu $g_i \neq g_j$ pentru orice $i \neq j$ devine $c'=(g_1', g_2', \dots, g_L')$, unde $g_i, g_i' \in \{val_1, val_2, \dots, val_L\}$, pt. $i=1, 2, \dots, L$ a.î. $g_i' \neq g_j'$ pentru orice $i \neq j$.

■ Mutație k-opt

■ Ideea de bază

- Se aleg 2 substringuri disjuncte și de lungime k
- Se interchimbă 2 elemente ale acestor substringuri de gene



Algoritmi evolutivi – algoritm

Proiectare – mutația (reperez. reală)



- Un cromozom $c=(g_1, g_2, \dots, g_L)$ devine $c'=(g_1', g_2', \dots, g_L')$, unde $g_i, g_i' \in [LI_i, LS_i]$, pt. $i=1, 2, \dots, L$

- Mutație uniformă
 - Ideea de bază
 - g_i' este schimbată cu probabilitatea p_m la o valoare aleasă aleator uniform din $[LI_i, LS_i]$

Algoritmi evolutivi – algoritm

Proiectare – mutația (reprez. reală)



- Un cromozom $c=(g_1, g_2, \dots, g_L)$ devine $c'=(g_1', g_2', \dots, g_L')$, unde $g_i, g_i' \in [LI_i, LS_i]$, pt. $i=1, 2, \dots, L$

- Mutație neuniformă
 - Ideea de bază
 - Valoarea unei gene este schimbată (cu probabilitatea p_m) prin adăugarea unei valori (pozitivă sau negativă)
 - valoarea \rightarrow face parte dintr-o distribuție
 - $N(\mu, \sigma)$ (Gaussiană) cu $\mu = 0$
 - Cauchy (x_0, γ)
 - Laplace (μ, b)
 - și readusă la $[LI_i, LS_i]$ (dacă este necesar) – *clamping*

Algoritmi evolutivi – algoritm Proiectare - recombinaarea

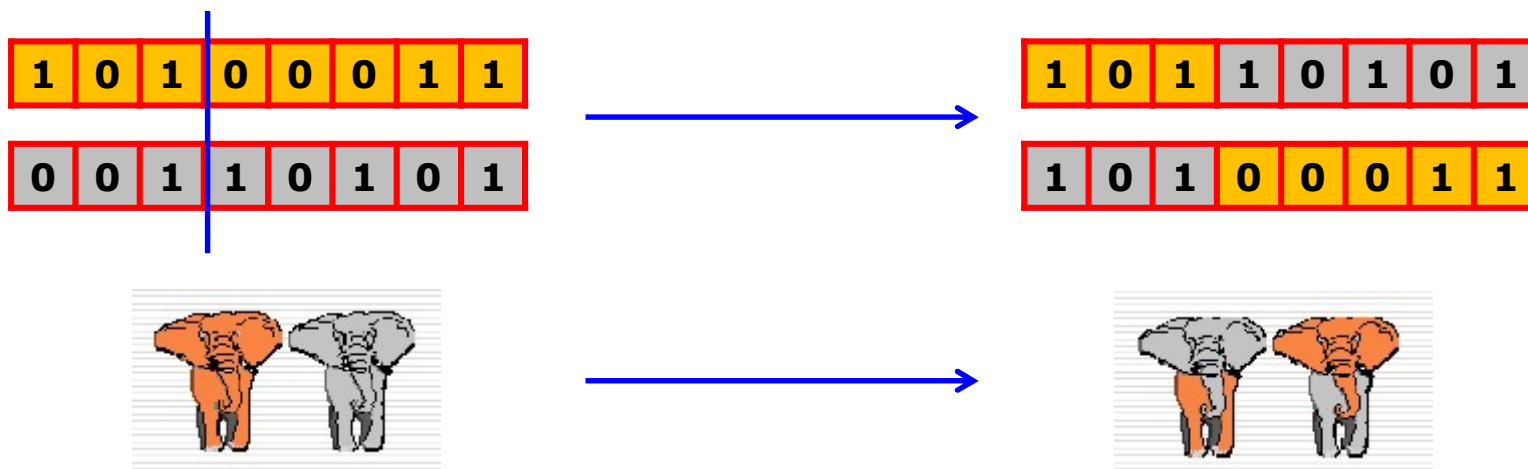


□ Scop

- Amestecarea informațiilor preluate din părinți

□ Proprietăți

- Descendentul trebuie să moștenească ceva de la fiecare dintre părinți
 - Alegerea informațiilor care se amestecă este aleatoare
- Operator de exploatare probabilistică (p_c) a spațiilor deja descoperite
- Descendenții pot să fie mai buni, la fel de buni sau mai slabi decât părinții lor
- Efectele sale se reduc pe măsură ce căutarea converge



Algoritmi evolutivi – algoritm

Proiectare - recombinarea



□ Tipologie – în funcție de reprezentarea indivizilor

■ Reprezentare binară și întreagă

- Cu puncte de tăietură
- Uniformă

■ Reprezentare cu permutări

- Încrucișare prin ordonare (versiunea 1 și versiunea 2)
- Încrucișare transformată parțial (Partially Mapped Crossover)
- Încrucișare ciclică
- Încrucișare bazată pe legături (muchii)

■ Reprezentare reală

- Discretă
- Intermediară (aritmetică)
 - Aritmetică singulară
 - Aritmetică simplă
 - Aritmetică completă
- Geometrică
- Încrucișare amestecată
- Încrucișare binară simulată

■ Reprezentare cu arbori

- Încrucișare de sub-arbori → într-un curs viitor

Algoritmi evolutivi – algoritm

Proiectare – recombinarea (reprez. binară și întreagă)

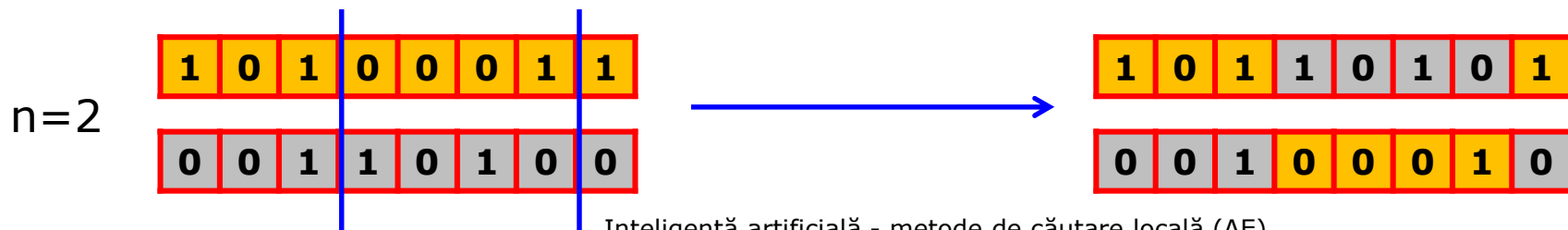


- Din 2 cromozomi părinți
 - $p_1 = (g_1^1, g_2^1, \dots, g_L^1)$ și $p_2 = (g_1^2, g_2^2, \dots, g_L^2)$
- se obțin 2 descendenți
 - $c_1 = (g_1', g_2', \dots, g_L')$ și $c_2 = (g_1'', g_2'', \dots, g_L'')$,
 - unde $g_i^1, g_i^2, g_i', g_i'' \in \{0, 1\} / \{val_1, val_2, \dots, val_k\}$, pt. $i=1, 2, \dots, L$

□ Încrucișare cu n puncte de tăietură

■ Ideea de bază

- Se aleg n puncte de tăietură ($n < L$)
- Se taie cromozomii părinți prin aceste puncte
- Se lipesc părțile obținute, alternând părinții



Algoritmi evolutivi – algoritm

Proiectare – recombinaarea (reprez. binară și întreagă)



□ Încrucișare cu n puncte de tăietură

■ Proprietăți

- Media valorilor codate de părinți = media valorilor codate de descendenți

- Ex. Reprezentarea binară pe 4 biți a numerelor întregi – XO cu $n = 1$ după bitul 2

- $p_1 = (1,0,1,0)$, $p_2 = (1,1,0,1)$
- $d_1 = (1,0, 0,1)$, $d_2 = (1,1,1,0)$
- $val(p_1) = 10$, $val(p_2) = 13 \rightarrow (val(p_1) + val(p_2))/2 = 23/2=11.5$
- $val(d_1) = 9$, $val(d_2) = 14 \rightarrow (val(d_1) + val(d_2))/2 = 23/2=11.5$

- Ex. Reprezentare binară pe 4 biți pentru problema rucsacului de capacitate $K = 10$ cu 4 obiecte de greutate și valoare $((2,7), (1,8), (3,1), (2,3))$

- $p_1 = (1,0,1,0)$, $p_2 = (1,1,0,1)$
- $d_1 = (1,0, 0,1)$, $d_2 = (1,1,1,0)$
- $val(p_1) = 8$, $val(p_2) = 18 \rightarrow (val(p_1) + val(p_2))/2 = 26/2=13$
- $val(d_1) = 10$, $val(d_2) = 16 \rightarrow (val(d_1) + val(d_2))/2 = 26/2=13$

- Probabilitatea apariției unui factor de răspândire $\beta \approx 1$ este mai mare decât probabilitatea oricărui alt factor

$$\beta = \frac{|val(d_1) - val(d_2)|}{|val(p_1) - val(p_2)|}$$

- Încrucișare prin contracție $\beta < 1$
 - Valorile descendenților se află între valorile părinților
- Încrucișare prin extensie $\beta > 1$
 - Valorile părinților se află între valorile descendenților
- Încrucișare staționară $\beta = 1$
 - Valorile descendenților coincid cu valorile părinților

Algoritmi evolutivi – algoritm

Proiectare – recombinarea (reprez. binară și întreagă)



□ Din 2 cromozomi părinți

- $p_1 = (g_1^1, g_2^1, \dots, g_L^1)$ și $p_2 = (g_1^2, g_2^2, \dots, g_L^2)$

□ se obțin 2 descendenți

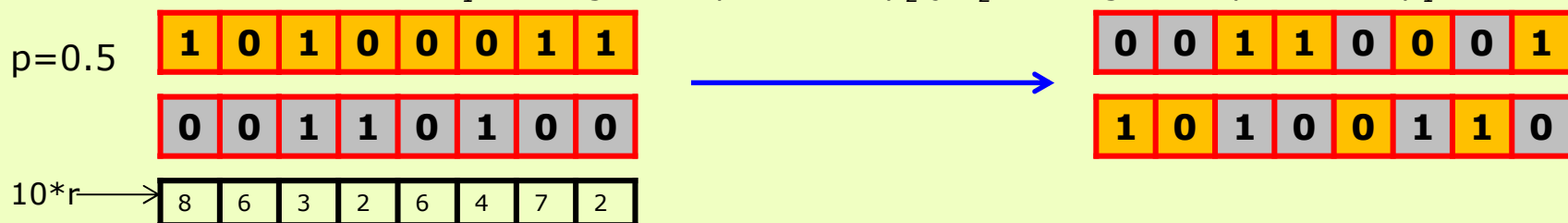
- $c_1 = (g_1', g_2', \dots, g_L')$ și $c_2 = (g_1'', g_2'', \dots, g_L'')$,
- unde $g_i^1, g_i^2, g_i', g_i'' \in \{0, 1\} / \{val_1, val_2, \dots, val_k\}$, pt. $i=1, 2, \dots, L$

□ Încrucișare uniformă

■ Ideea de bază

- Fiecare genă a unui descendent provine dintr-un părinte ales aleator și uniform:

- Pentru fiecare genă în parte se generează un număr aleator r care respectă legea uniformă
- Dacă numărul generat $r < probabilitatea p$ (de obicei $p=0.5$), c_1 va lua gena respectivă din p_1 și c_2 va lua gena respectivă din p_2 ,
- Altfel c_1 va lua gena respectivă din p_2 și c_2 va lua gena respectivă din p_1

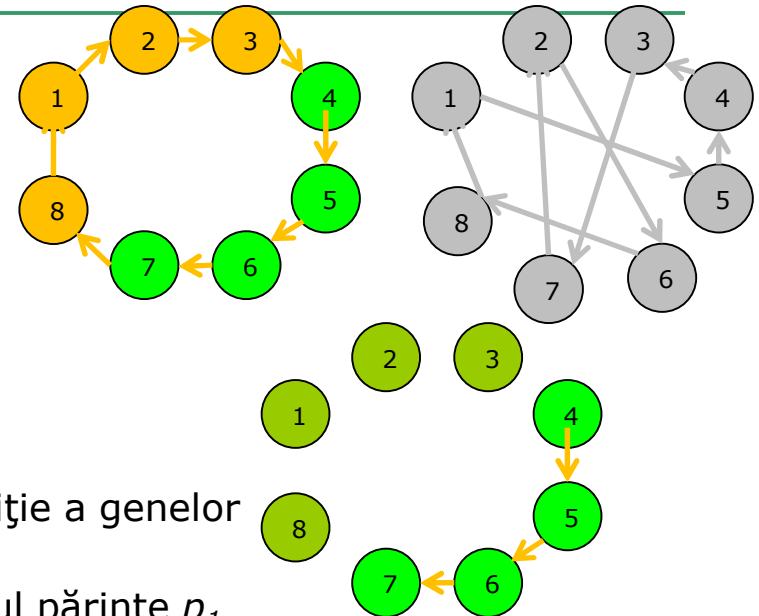


Algoritmi evolutivi – algoritm

Proiectare – recombinarea (reprez. permutare)



- Din 2 cromozomi părinți
 - $p_1 = (g_1^1, g_2^1, \dots, g_L^1)$ și $p_2 = (g_1^2, g_2^2, \dots, g_L^2)$
- se obțin 2 descendenți
 - $c_1 = (g_1', g_2', \dots, g_L')$ și $c_2 = (g_1'', g_2'', \dots, g_L'')$,
 - unde $g_i^1, g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, pt. $i=1, 2, \dots, L$
- Încrucișare ordonată
 - Ideea de bază
 - Descendenții păstrează ordinea de apariție a genelor părinților
 - Se alege un substring de gene din primul părinte p_1
 - Se copiază substringul din p_1 în descendentul d_1 (pe poziții corespondente)



Algoritmi evolutivi – algoritm

Proiectare – recombinaarea (reprez. permutare)

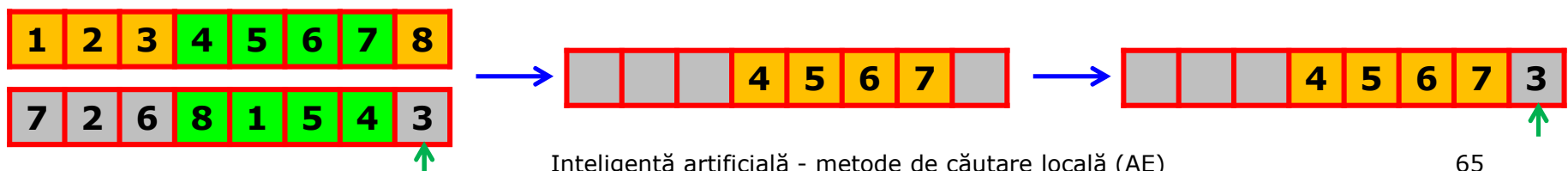
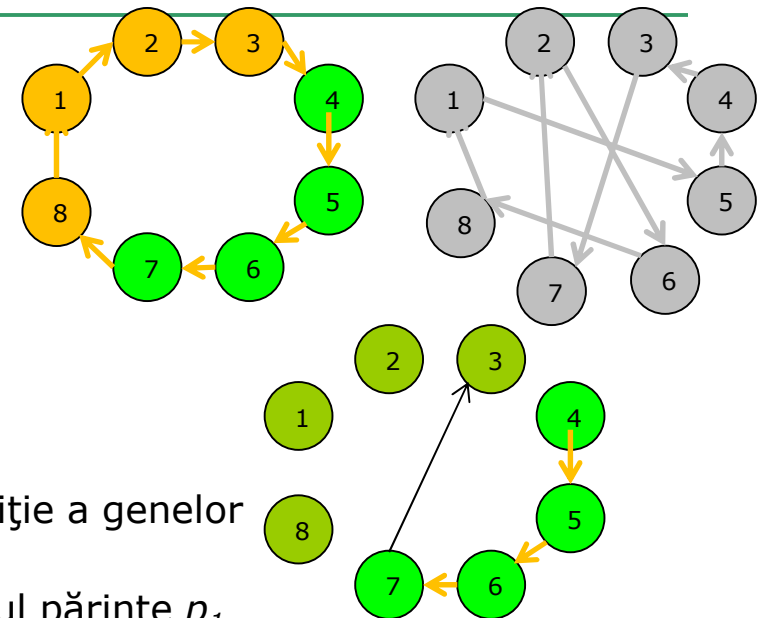


- Din 2 cromozomi părinți
 - $p_1 = (g_1^1, g_2^1, \dots, g_L^1)$ și $p_2 = (g_1^2, g_2^2, \dots, g_L^2)$
- se obțin 2 descendenți
 - $c_1 = (g_1', g_2', \dots, g_L')$ și $c_2 = (g_1'', g_2'', \dots, g_L'')$,
 - unde $g_i^1, g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, pt. $i=1, 2, \dots, L$

□ Încrucișare ordonată

■ Ideea de bază

- Descendenții păstrează ordinea de apariție a genelor părinților
- Se alege un substring de gene din primul părinte p_1
- Se copiază substringul din p_1 în descendentul d_1 (pe poziții corespondente)
- Se copiază genele din p_2 în descendentul d_1 astfel:
 - Începând cu prima poziție de după terminarea substringului
 - Respectând ordinea genelor din p_2 și
 - Re-luând genele de la prima poziție (dacă s-a ajuns la sfârșit)

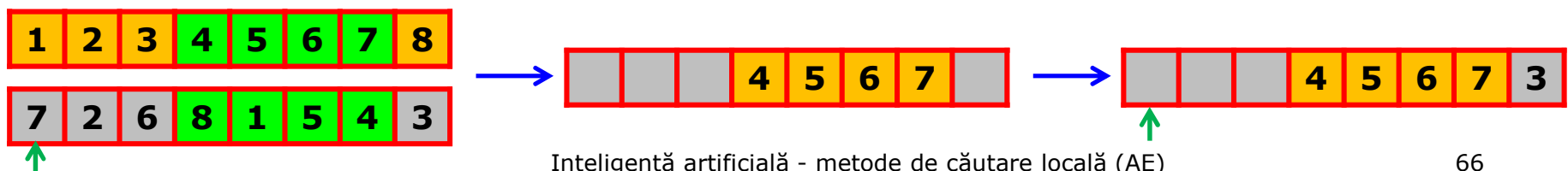
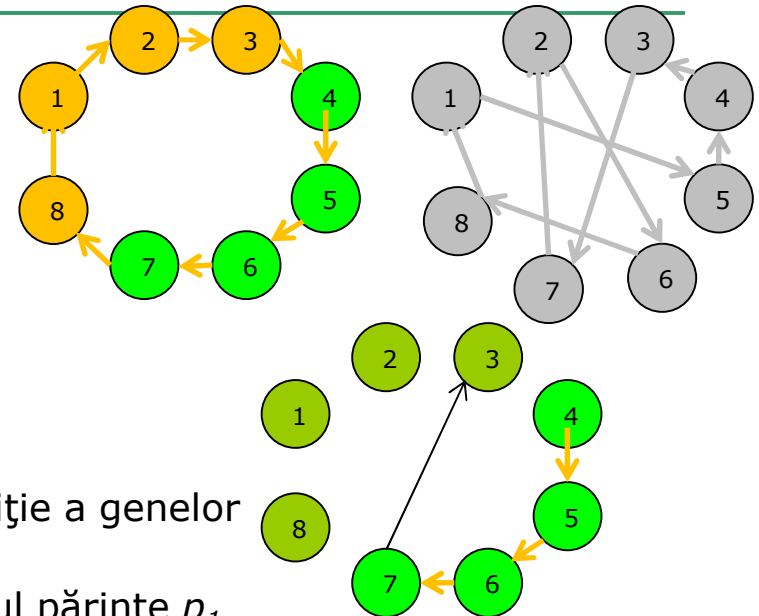


Algoritmi evolutivi – algoritm

Proiectare – recombinarea (reprez. permutare)



- Din 2 cromozomi părinți
 - $p_1 = (g_1^1, g_2^1, \dots, g_L^1)$ și $p_2 = (g_1^2, g_2^2, \dots, g_L^2)$
- se obțin 2 descendenți
 - $c_1 = (g_1', g_2', \dots, g_L')$ și $c_2 = (g_1'', g_2'', \dots, g_L'')$,
 - unde $g_i^1, g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, pt. $i=1, 2, \dots, L$
- Încrucișare ordonată
 - Ideea de bază
 - Descendenții păstrează ordinea de apariție a genelor părinților
 - Se alege un substring de gene din primul părinte p_1
 - Se copiază substringul din p_1 în descendentul d_1 (pe poziții corespondente)
 - Se copiază genele din p_2 în descendentul d_1 astfel:
 - Începând cu prima poziție de după terminarea substringului
 - Respectând ordinea genelor din p_2 și
 - Re-luând genele de la prima poziție (dacă s-a ajuns la sfârșit)



Algoritmi evolutivi – algoritm

Proiectare – recombinarea (reprez. permutare)

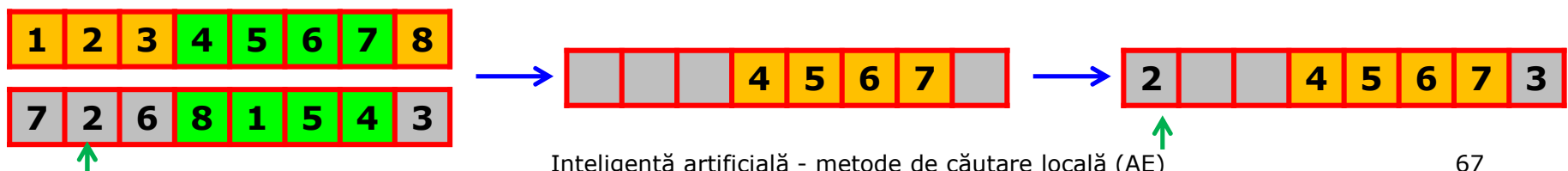
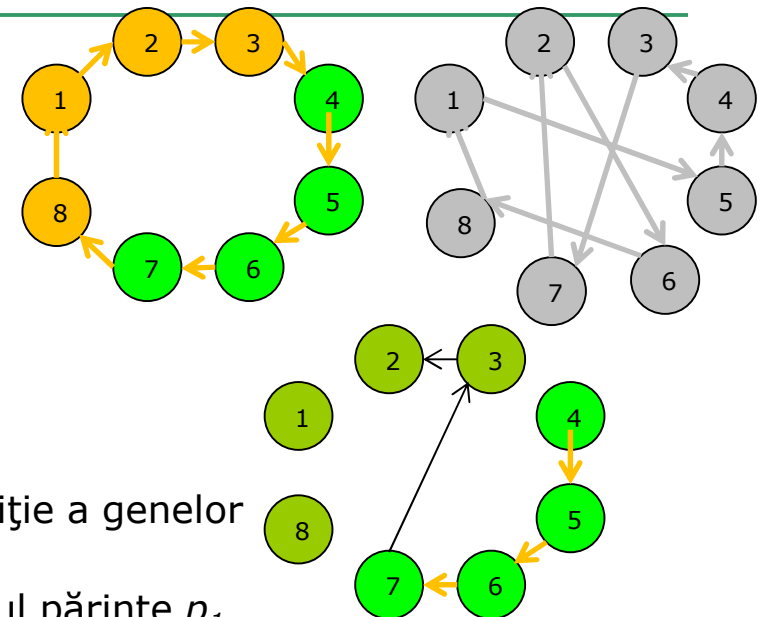


- Din 2 cromozomi părinți
 - $p_1 = (g_1^1, g_2^1, \dots, g_L^1)$ și $p_2 = (g_1^2, g_2^2, \dots, g_L^2)$
- se obțin 2 descendenți
 - $c_1 = (g_1', g_2', \dots, g_L')$ și $c_2 = (g_1'', g_2'', \dots, g_L'')$,
 - unde $g_i^1, g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, pt. $i=1, 2, \dots, L$

□ Încrucișare ordonată

■ Ideea de bază

- Descendenții păstrează ordinea de apariție a genelor părinților
- Se alege un substring de gene din primul părinte p_1
- Se copiază substringul din p_1 în descendentul d_1 (pe poziții corespondente)
- Se copiază genele din p_2 în descendentul d_1 astfel:
 - Începând cu prima poziție de după terminarea substringului
 - Respectând ordinea genelor din p_2 și
 - Re-luând genele de la prima poziție (dacă s-a ajuns la sfârșit)



Algoritmi evolutivi – algoritm

Proiectare – recombinarea (reprez. permutare)

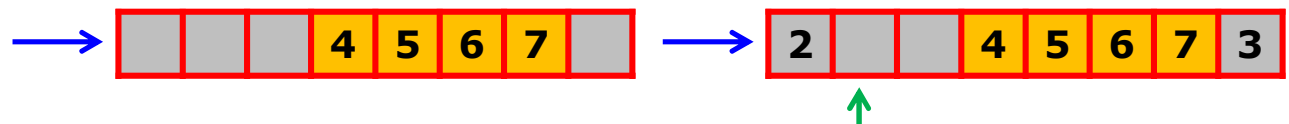
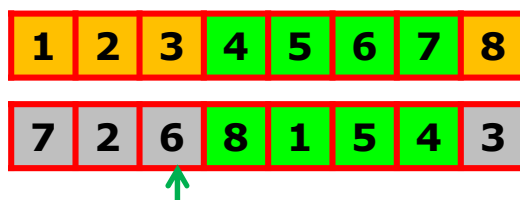
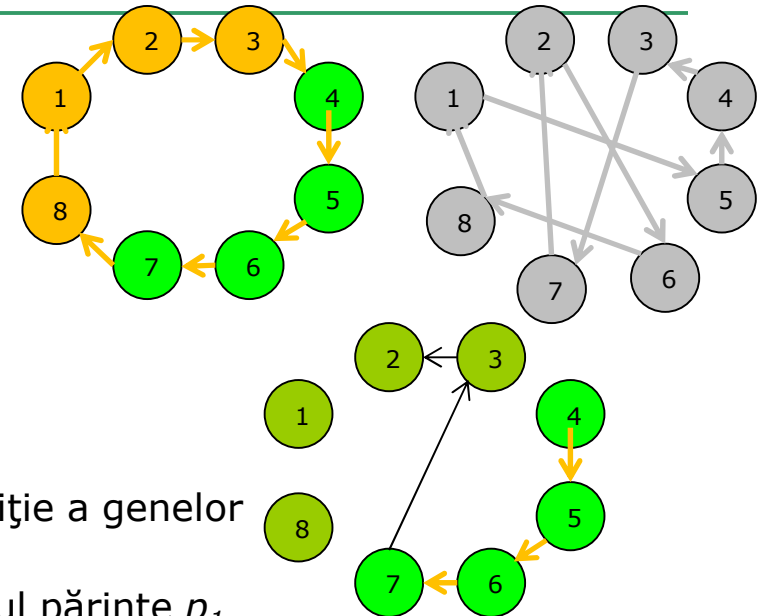


- Din 2 cromozomi părinți
 - $p_1 = (g_1^1, g_2^1, \dots, g_L^1)$ și $p_2 = (g_1^2, g_2^2, \dots, g_L^2)$
- se obțin 2 descendenți
 - $c_1 = (g_1', g_2', \dots, g_L')$ și $c_2 = (g_1'', g_2'', \dots, g_L'')$,
 - unde $g_i^1, g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, pt. $i=1, 2, \dots, L$

□ Încrucișare ordonată

■ Ideea de bază

- Descendenții păstrează ordinea de apariție a genelor părinților
- Se alege un substring de gene din primul părinte p_1
- Se copiază substringul din p_1 în descendentul d_1 (pe poziții corespondente)
- Se copiază genele din p_2 în descendentul d_1 astfel:
 - Începând cu prima poziție de după terminarea substringului
 - Respectând ordinea genelor din p_2 și
 - Re-luând genele de la prima poziție (dacă s-a ajuns la sfârșit)



Algoritmi evolutivi – algoritm

Proiectare – recombinarea (reprez. permutare)

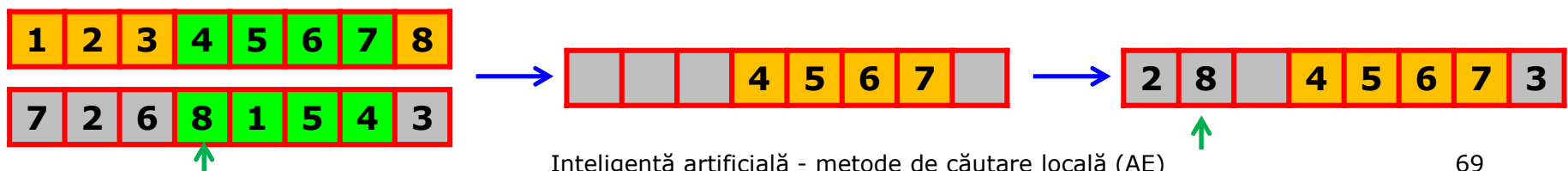
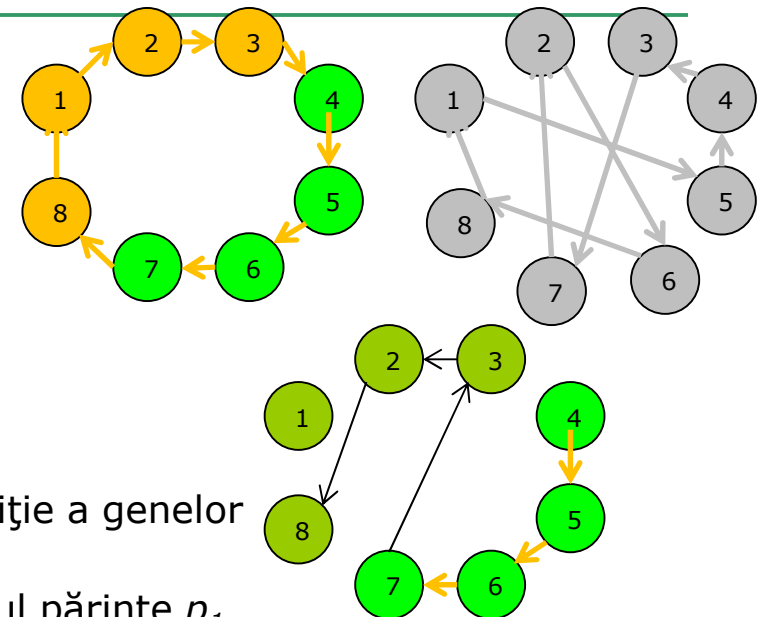


- Din 2 cromozomi părinți
 - $p_1 = (g_1^1, g_2^1, \dots, g_L^1)$ și $p_2 = (g_1^2, g_2^2, \dots, g_L^2)$
- se obțin 2 descendenți
 - $c_1 = (g_1', g_2', \dots, g_L')$ și $c_2 = (g_1'', g_2'', \dots, g_L'')$,
 - unde $g_i^1, g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, pt. $i=1, 2, \dots, L$

□ Încrucișare ordonată

■ Ideea de bază

- Descendenții păstrează ordinea de apariție a genelor părinților
- Se alege un substring de gene din primul părinte p_1
- Se copiază substringul din p_1 în descendentul d_1 (pe poziții corespondente)
- Se copiază genele din p_2 în descendentul d_1 astfel:
 - Începând cu prima poziție de după terminarea substringului
 - Respectând ordinea genelor din p_2 și
 - Re-luând genele de la prima poziție (dacă s-a ajuns la sfârșit)



Algoritmi evolutivi – algoritm

Proiectare – recombinarea (reprez. permutare)



□ Din 2 cromozomi părinți

- $p_1 = (g_1^1, g_2^1, \dots, g_L^1)$ și $p_2 = (g_1^2, g_2^2, \dots, g_L^2)$

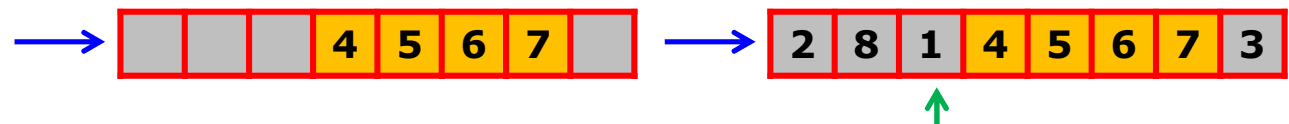
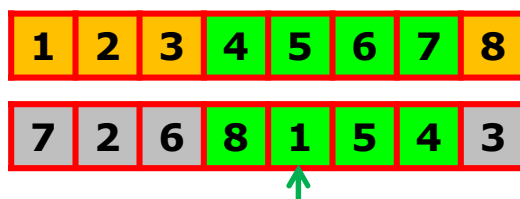
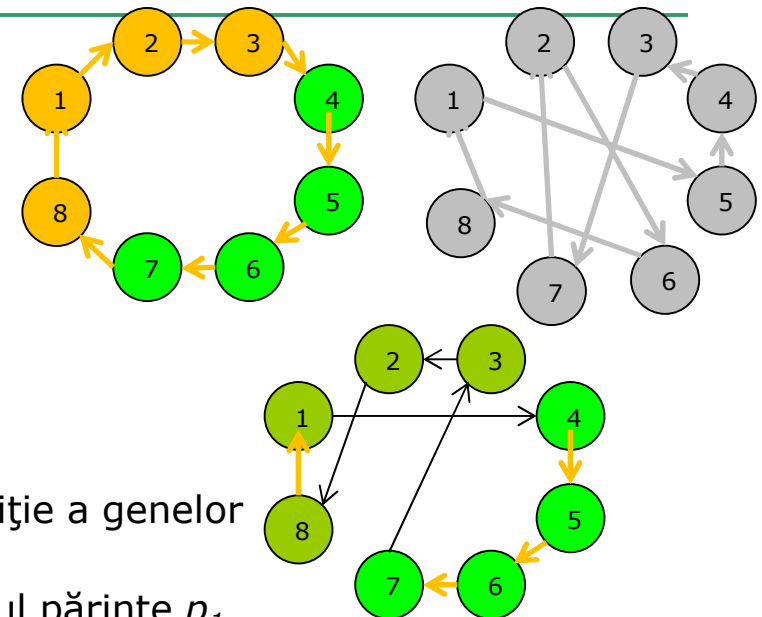
□ se obțin 2 descendenți

- $c_1 = (g_1', g_2', \dots, g_L')$ și $c_2 = (g_1'', g_2'', \dots, g_L'')$,
- unde $g_i^1, g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, pt. $i=1, 2, \dots, L$

□ Încrucișare ordonată

■ Ideea de bază

- Descendenții păstrează ordinea de apariție a genelor părinților
- Se alege un substring de gene din primul părinte p_1
- Se copiază substringul din p_1 în descendentul d_1 (pe poziții corespondente)
- Se copiază genele din p_2 în descendentul d_1 astfel:
 - Începând cu prima poziție de după terminarea substringului
 - Respectând ordinea genelor din p_2 și
 - Re-luând genele de la prima poziție (dacă s-a ajuns la sfârșit)



Algoritmi evolutivi – algoritm

Proiectare – recombinarea (reprez. permutare)

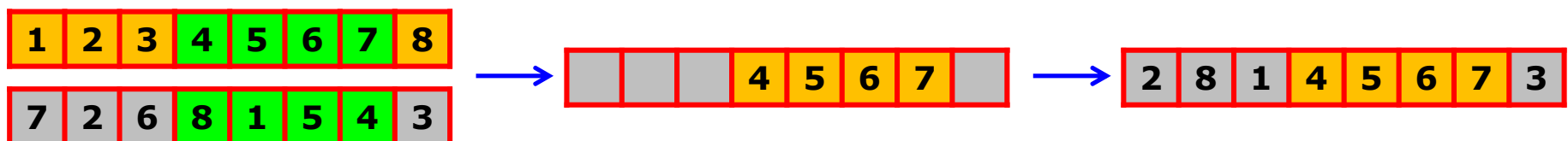
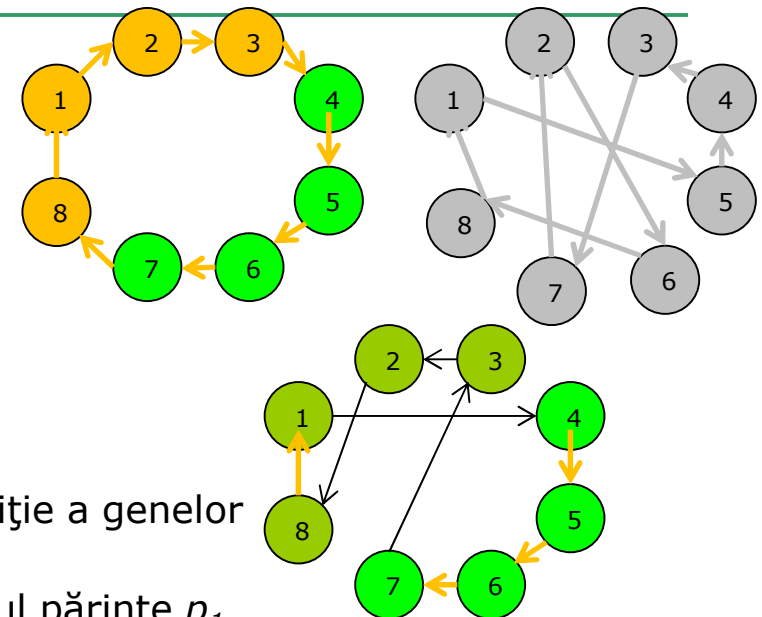


- Din 2 cromozomi părinți
 - $p_1 = (g_1^1, g_2^1, \dots, g_L^1)$ și $p_2 = (g_1^2, g_2^2, \dots, g_L^2)$
- se obțin 2 descendenți
 - $c_1 = (g_1', g_2', \dots, g_L')$ și $c_2 = (g_1'', g_2'', \dots, g_L'')$,
 - unde $g_i^1, g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, pt. $i=1, 2, \dots, L$

□ Încrucișare ordonată

■ Ideea de bază

- Descendenții păstrează ordinea de apariție a genelor părinților
- Se alege un substring de gene din primul părinte p_1
- Se copiază substringul din p_1 în descendentul d_1 (pe poziții corespondente)
- Se copiază genele din p_2 în descendentul d_1 astfel:
 - Începând cu prima poziție de după terminarea substringului
 - Respectând ordinea genelor din p_2 și
 - Re-luând genele de la prima poziție (dacă s-a ajuns la sfârșit)
- Se reia procedeul pentru al doilea descendent d_2 .

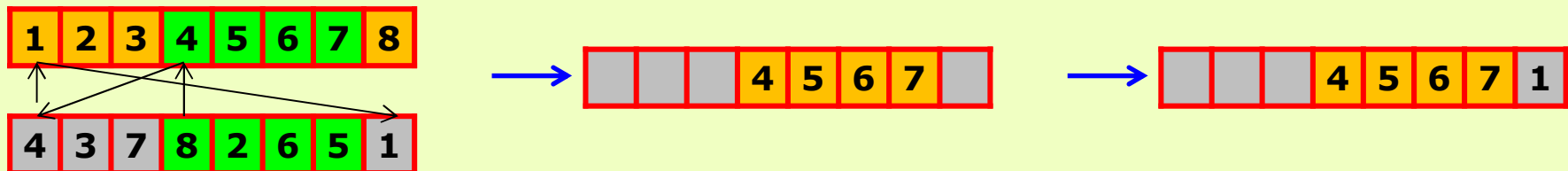


Algoritmi evolutivi – algoritm

Proiectare – recombinarea (reprez. permutare)



- Din 2 cromozomi părinți
 - $p_1 = (g_1^1, g_2^1, \dots, g_L^1)$ și $p_2 = (g_1^2, g_2^2, \dots, g_L^2)$
- se obțin 2 descendenți
 - $c_1 = (g_1', g_2', \dots, g_L')$ și $c_2 = (g_1'', g_2'', \dots, g_L'')$,
 - unde $g_i^1, g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, pt. $i=1, 2, \dots, L$
- Încrucișare parțial transformată (*partially mapped XO*)
 - Ideea de bază
 - Se alege un substring de gene din primul părinte p_1
 - Se copiază substringul din p_1 în descendentul d_1 (pe poziții corespondente)
 - Se iau pe rând elementele i din substringul din p_2 care nu apar în substringul din p_1 și se determină care element j a fost copiat în locul lui din p_1
 - Se plasează i în d_1 în poziția ocupată de j în p_2 (dacă locul este liber)
 - Dacă locul ocupat de j în p_2 a fost deja completat în d_1 cu elementul k , i se pune în locul ocupat de k în p_2
 - Restul elementelor se copiază din p_2 în d_1
 - Pentru descendentul d_2 se procedează similar, dar inversând părinții

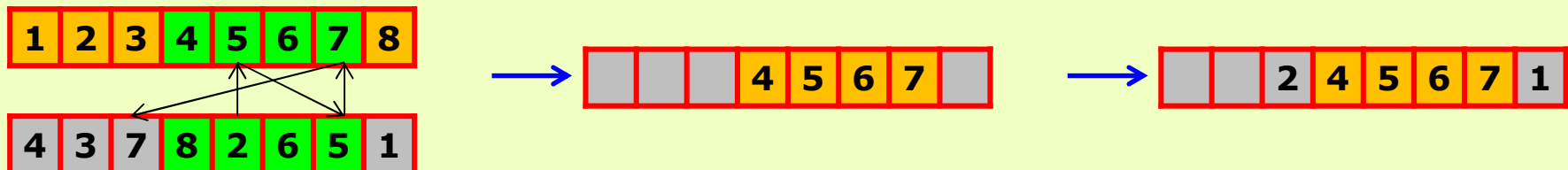


Algoritmi evolutivi – algoritm

Proiectare – recombinarea (reprez. permutare)



- Din 2 cromozomi părinți
 - $p_1 = (g_1^1, g_2^1, \dots, g_L^1)$ și $p_2 = (g_1^2, g_2^2, \dots, g_L^2)$
- se obțin 2 descendenți
 - $c_1 = (g_1', g_2', \dots, g_L')$ și $c_2 = (g_1'', g_2'', \dots, g_L'')$,
 - unde $g_i^1, g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, pt. $i=1, 2, \dots, L$
- Încrucișare parțial transformată (*partially mapped XO*)
 - Ideea de bază
 - Se alege un substring de gene din primul părinte p_1
 - Se copiază substringul din p_1 în descendentul d_1 (pe poziții corespondente)
 - Se iau pe rând elementele i din substringul din p_2 care nu apar în substringul din p_1 și se determină care element j a fost copiat în locul lui din p_1
 - Se plasează i în d_1 în poziția ocupată de j în p_2 (dacă locul este liber)
 - Dacă locul ocupat de j în p_2 a fost deja completat în d_1 cu elementul k , i se pune în locul ocupat de k în p_2
 - Restul elementelor se copiază din p_2 în d_1
 - Pentru descendentul d_2 se procedează similar, dar inversând părinții



Algoritmi evolutivi – algoritm

Proiectare – recombinarea (reprez. permutare)



□ Din 2 cromozomi părinți

- $p_1 = (g_1^1, g_2^1, \dots, g_L^1)$ și $p_2 = (g_1^2, g_2^2, \dots, g_L^2)$

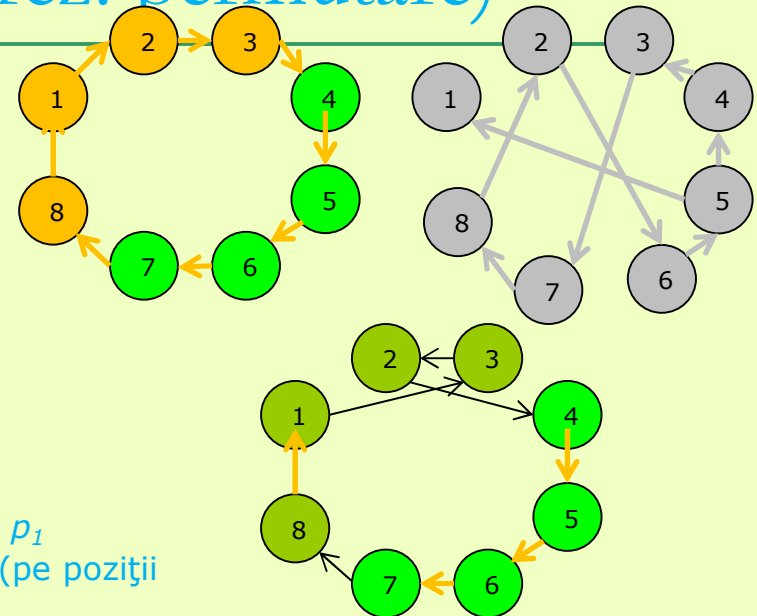
□ se obțin 2 descendenți

- $c_1 = (g_1', g_2', \dots, g_L')$ și $c_2 = (g_1'', g_2'', \dots, g_L'')$,
- unde $g_i^1, g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, pt. $i=1, 2, \dots, L$

□ Încrucișare parțial transformată

■ Ideea de bază

- Se alege un substring de gene din primul părinte p_1
- Se copiază substringul din p_1 în descendentul d_1 (pe poziții corespondente)
- Se iau pe rând elementele i din substringul din p_2 care nu apar în substringul din p_1 și se determină care element j a fost copiat în locul lui i din p_1
- Se plasează i în d_1 în poziția ocupată de j în p_2 (dacă locul este liber)
- Dacă locul ocupat de j în p_2 a fost deja completat în d_1 cu elementul k , i se pune în locul ocupat de k în p_2
- Restul elementelor se copiază din p_2 în d_1
- Pentru descendentul d_2 se procedează similar, dar inversând părinții



1 2 3 4 5 6 7 8

4 3 7 8 2 6 5 1



4 5 6 7



8 3 2 4 5 6 7 1

Algoritmi evolutivi – algoritm

Proiectare – recombinarea (reprez. permutare)



□ Din 2 cromozomi părinți

- $p_1 = (g_1^1, g_2^1, \dots, g_L^1)$ și $p_2 = (g_1^2, g_2^2, \dots, g_L^2)$

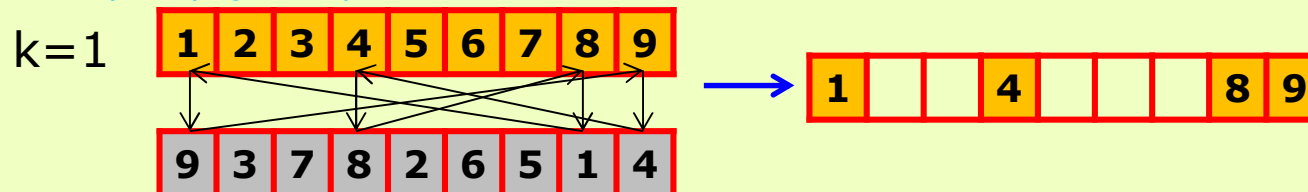
□ se obțin 2 descendenți

- $c_1 = (g_1', g_2', \dots, g_L')$ și $c_2 = (g_1'', g_2'', \dots, g_L'')$,
- unde $g_i^1, g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, pt. $i=1, 2, \dots, L$

□ Încrucișare ciclică

■ Ideea de bază

1. inițial $k = 1$
2. Se formează un ciclu
 - Se adaugă în ciclu gena de pe poziția k din p_1 (g_k^1)
 - Se consideră gena de pe poziția k din p_2 (g_k^2)
 - Se alege gena din p_1 cu valoarea egală cu g_k^2 (g_r^1) și se include în ciclu
 - Se consideră gena de pe poziția r din p_2 (g_r^2)
 - Se repetă pașii anteriori până când se ajunge la gena de pe poziția k din p_1
3. Se copiază genele din ciclu în d_1 (respectând pozițiile pe care apar în p_1)
4. Se incrementează k și se formează un nou ciclu dar cu genele din p_2
5. Se copiază genele din ciclu în d_1 (respectând pozițiile pe care apar în p_2)
6. Se repetă pașii 2-5 până când $k = L$



Algoritmi evolutivi – algoritm

Proiectare – recombinarea (reprez. permutare)



□ Din 2 cromozomi părinți

- $p_1 = (g_1^1, g_2^1, \dots, g_L^1)$ și $p_2 = (g_1^2, g_2^2, \dots, g_L^2)$

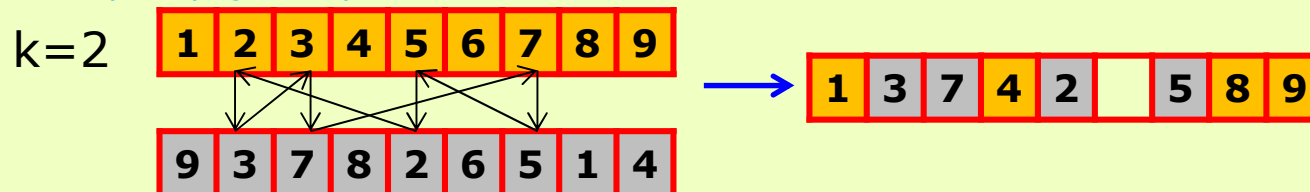
□ se obțin 2 descendenți

- $c_1 = (g_1', g_2', \dots, g_L')$ și $c_2 = (g_1'', g_2'', \dots, g_L'')$,
- unde $g_i^1, g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, pt. $i=1, 2, \dots, L$

□ Încrucișare ciclică

■ Ideea de bază

1. inițial $k = 1$
2. Se formează un ciclu
 - Se adaugă în ciclu gena de pe poziția k din p_1 (g_k^1)
 - Se consideră gena de pe poziția k din p_2 (g_k^2)
 - Se alege gena din p_1 cu valoarea egală cu g_k^2 (g_r^1) și se include în ciclu
 - Se consideră gena de pe poziția r din p_2 (g_r^2)
 - Se repetă pașii anteriori până când se ajunge la gena de pe poziția k din p_1
3. Se copiază genele din ciclu în d_1 (respectând pozițiile pe care apar în p_1)
4. Se incrementează k și se formează un nou ciclu dar cu genele din p_2
5. Se copiază genele din ciclu în d_1 (respectând pozițiile pe care apar în p_2)
6. Se repetă pașii 2-5 până când $k = L$



Algoritmi evolutivi – algoritm

Proiectare – recombinarea (reprez. permutare)



□ Din 2 cromozomi părinți

- $p_1 = (g_1^1, g_2^1, \dots, g_L^1)$ și $p_2 = (g_1^2, g_2^2, \dots, g_L^2)$

□ se obțin 2 descendenți

- $c_1 = (g_1', g_2', \dots, g_L')$ și $c_2 = (g_1'', g_2'', \dots, g_L'')$,
- unde $g_i^1, g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, pt. $i=1, 2, \dots, L$

□ Încrucișare ciclică

■ Ideea de bază

1. inițial $k = 1$
2. Se formează un ciclu
 - Se adaugă în ciclu gena de pe poziția k din p_1 (g_k^1)
 - Se consideră gena de pe poziția k din p_2 (g_k^2)
 - Se alege gena din p_1 cu valoarea egală cu g_k^2 (g_r^1) și se include în ciclu
 - Se consideră gena de pe poziția r din p_2 (g_r^2)
 - Se repetă pașii anteriori până când se ajunge la gena de pe poziția k din p_1
3. Se copiază genele din ciclu în d_1 (respectând pozițiile pe care apar în p_1)
4. Se incrementează k și se formează un nou ciclu dar cu genele din p_2
5. Se copiază genele din ciclu în d_1 (respectând pozițiile pe care apar în p_2)
6. Se repetă pașii 2-5 până când $k = L$

$k=3$



Algoritmi evolutivi – algoritm

Proiectare – recombinarea (reprez. permutare)



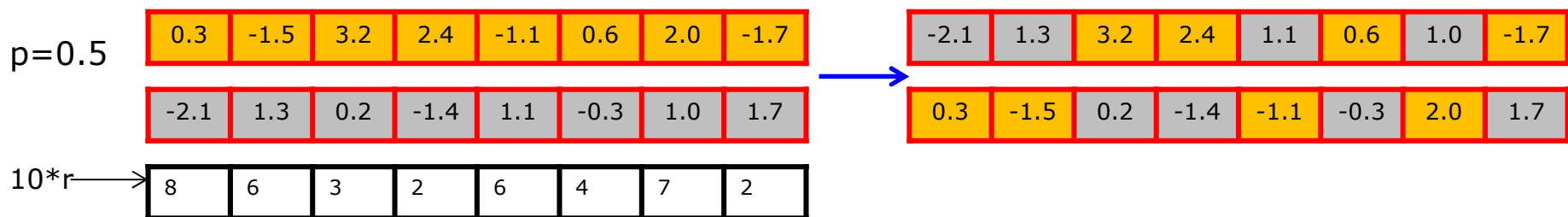
- Din 2 cromozomi părinți
 - $p_1 = (g_1^1, g_2^1, \dots, g_L^1)$ și $p_2 = (g_1^2, g_2^2, \dots, g_L^2)$
- se obțin 2 descendenți
 - $c_1 = (g_1', g_2', \dots, g_L')$ și $c_2 = (g_1'', g_2'', \dots, g_L'')$,
 - unde $g_i^1, g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, pt. $i=1, 2, \dots, L$
- Încrucișare bazată pe muchii
 - A se consulta: *Whitley, Darrell, Timothy Starkweather, D'Ann Fuquay (1989). "Scheduling problems and traveling salesman: The genetic edge recombination operator". International Conference on Genetic Algorithms. pp. 133–140 [link](#)*

Algoritmi evolutivi – algoritm

Proiectare – recombinarea (reperez. reală)



- Din 2 cromozomi părinți
 - $p_1 = (g_1^1, g_2^1, \dots, g_L^1)$ și $p_2 = (g_1^2, g_2^2, \dots, g_L^2)$
- se obțin 2 descendenți
 - $c_1 = (g_1', g_2', \dots, g_L')$ și $c_2 = (g_1'', g_2'', \dots, g_L'')$,
 - unde $g_i^1, g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, pt. $i=1, 2, \dots, L$
- Încrucișare discretă
 - Ideea de bază
 - Fiecare genă a unui descendent este luată (cu aceeași probabilitate, $p = 0.5$) dintr-unul din părinți
 - Similar încrucișării uniforme de la reprezentarea binară/întreagă
 - Nu se modifică valorile efective ale genelor (nu se creează informație nouă)



Algoritmi evolutivi – algoritm

Proiectare – recombinaarea (reperez. reală)



- Din 2 cromozomi părinți
 - $p_1 = (g_1^1, g_2^1, \dots, g_L^1)$ și $p_2 = (g_1^2, g_2^2, \dots, g_L^2)$
- se obțin 2 descendenți
 - $c_1 = (g_1', g_2', \dots, g_L')$ și $c_2 = (g_1'', g_2'', \dots, g_L'')$,
unde $g_i^1, g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, pt. $i=1, 2, \dots, L$
- Încrucișare intermediară (aritmetică)
 - Ideea de bază
 - Se creează copii aflați (ca valoare) între părinți → încrucișare aritmetică
 - $z_i = \alpha x_i + (1 - \alpha) y_i$ unde $\alpha : 0 \leq \alpha \leq 1$.
 - Parametrul α poate fi:
 - Constant → încrucișare aritmetică uniformă
 - Variabil → ex. dependent de vârsta populației
 - Aleator pt fiecare încrucișare produsă
 - Apar noi valori ale genelor

■ Tipologie

- Încrucișare aritmetică singulară
- Încrucișare aritmetică simplă
- Încrucișare aritmetică completă

Algoritmi evolutivi – algoritm

Proiectare – recombinarea (reprez. reală)



- Din 2 cromozomi părinți
 - $p_1 = (g_1^1, g_2^1, \dots, g_L^1)$ și $p_2 = (g_1^2, g_2^2, \dots, g_L^2)$
- se obțin 2 descendenți
 - $c_1 = (g_1', g_2', \dots, g_L')$ și $c_2 = (g_1'', g_2'', \dots, g_L'')$,
 - unde $g_i^1, g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, pt. $i=1,2,\dots,L$
- Încrucișare intermediară (aritmetică) singulară
 - Se alege câte o genă (de același index k) din cei doi părinți și se combină
 - $g_k' = \alpha g_k^1 + (1-\alpha)g_k^2$
 - $g_k'' = (1-\alpha)g_k^1 + \alpha g_k^2$
 - Restul genelor rămân neschimbate
 - $g_i' = g_i^1$
 - $g_i'' = g_i^2$, pentru $i = 1, 2, \dots, L$ și $i \neq k$

$[LI, LS] = [-2.5, +3]$

$k=3$

$\alpha = 0.6$

0.3	-1.5	3.2	2.4	-1.1	0.6	2.0	-1.7
-2.1	1.3	0.2	-1.4	1.1	-0.3	1.0	1.7



0.3	-1.5	2.0	2.4	-1.1	0.6	2.0	-1.7
-2.1	1.3	1.4	-1.4	1.1	-0.3	1.0	1.7

$$0.6 \cdot 3.2 + (1-0.6) \cdot 0.2 = 2.0$$

$$(1-0.6) \cdot 3.2 + 0.6 \cdot 0.2 = 1.4$$

Algoritmi evolutivi – algoritm

Proiectare – recombinarea (reprez. reală)



- Din 2 cromozomi părinți
 - $p_1 = (g_1^1, g_2^1, \dots, g_L^1)$ și $p_2 = (g_1^2, g_2^2, \dots, g_L^2)$
- se obțin 2 descendenți
 - $c_1 = (g_1', g_2', \dots, g_L')$ și $c_2 = (g_1'', g_2'', \dots, g_L'')$,
 - unde $g_i^1, g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, pt. $i=1, 2, \dots, L$
- Încrucișare intermediară (aritmetică) simplă
 - Se alege o poziție k și se combină toate genele de după acea poziție
 - $g_i' = \alpha g_i^1 + (1-\alpha)g_i^2$
 - $g_i'' = (1-\alpha)g_i^1 + \alpha g_i^2$, pentru $i=k, k+1, \dots, L$
 - Genele de pe poziții $< k$ rămân neschimbate
 - $g_i' = g_i^1$
 - $g_i'' = g_i^2$, pentru $i = 1, 2, \dots, k-1$

$[LI, LS] = [-2.5, +3]$

$k=6$

$\alpha = 0.6$

0.3	-1.5	3.2	2.4	-1.1	0.6	2.0	-1.7
-2.1	1.3	0.2	-1.4	1.1	-0.3	1.0	1.7



0.3	-1.5	3.2	2.4	-1.1	0.24	1.6	-0.34
-2.1	1.3	0.2	-1.4	1.1	0.06	1.4	0.34

$$0.6 \cdot 0.6 + (1-0.6) \cdot (-0.3) = 0.24$$

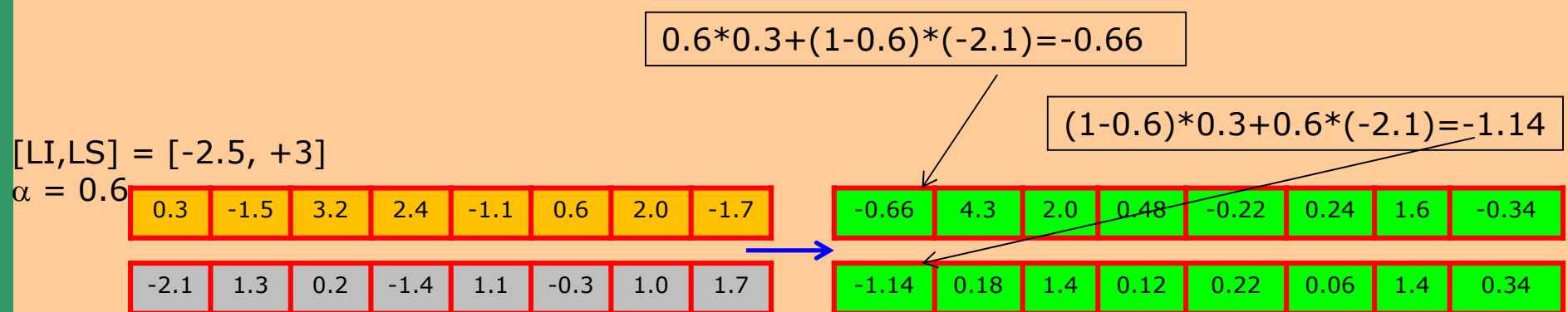
$$(1-0.6) \cdot 0.6 + 0.6 \cdot (-0.3) = 0.06$$

Algoritmi evolutivi – algoritm

Proiectare – recombinarea (reprez. reală)



- Din 2 cromozomi părinți
 - $p_1 = (g_1^1, g_2^1, \dots, g_L^1)$ și $p_2 = (g_1^2, g_2^2, \dots, g_L^2)$
- se obțin 2 descendenți
 - $c_1 = (g_1', g_2', \dots, g_L')$ și $c_2 = (g_1'', g_2'', \dots, g_L'')$,
 - unde $g_i^1, g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, pt. $i=1, 2, \dots, L$
- Încrucișare intermediară (aritmetică) completă
 - Toate genele (de pe poziții corespunzătoare) se combină
 - $g_i' = \alpha g_i^1 + (1-\alpha)g_i^2$
 - $g_i'' = (1-\alpha)g_i^1 + \alpha g_i^2$, pentru $i=1, 2, \dots, L$



Algoritmi evolutivi – algoritm

Proiectare – recombinarea (reperez. reală)



- Din 2 cromozomi părinți
 - $p_1 = (g_1^1, g_2^1, \dots, g_L^1)$ și $p_2 = (g_1^2, g_2^2, \dots, g_L^2)$
- se obțin 2 descendenți
 - $c_1 = (g_1', g_2', \dots, g_L')$ și $c_2 = (g_1'', g_2'', \dots, g_L'')$,
 - unde $g_i^1, g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, pt. $i=1, 2, \dots, L$

□ Încrucișare geometrică

■ Ideea de bază

- Fiecare genă a unui descendent reprezintă produsul genelor părinților, fiecare cu un anumit exponent ω , respectiv $1-\omega$ (unde ω număr real pozitiv subunitar)

$$g_i' = (g_i^1)^\omega (g_i^2)^{1-\omega}$$

$$g_i'' = (g_i^1)^{1-\omega} (g_i^2)^\omega$$

$$0.3^{0.7} + 2.1^{1-0.7} = 1.68$$

$$0.3^{1-0.7} + 2.1^{0.7} = 2.38$$

$[LI, LS] = [-2.5, +3]$

$\omega = 0.7$

0.3	1.5	3.2	2.4	1.1	0.6	2.0	1.7
-----	-----	-----	-----	-----	-----	-----	-----

2.1	1.3	0.2	1.4	1.1	0.3	1.0	1.7
-----	-----	-----	-----	-----	-----	-----	-----

1.68	2.41	2.87	2.95	2.10	1.40	2.62	2.62
------	------	------	------	------	------	------	------

2.38	2.33	1.74	2.57	2.10	1.29	2.23	2.62
------	------	------	------	------	------	------	------

Algoritmi evolutivi – algoritm

Proiectare – recombinarea (reperez. reală)



- Din 2 cromozomi părinți
 - $p_1 = (g_1^1, g_2^1, \dots, g_L^1)$ și $p_2 = (g_1^2, g_2^2, \dots, g_L^2)$
- se obține 1 descendent
 - $c_1 = (g_1', g_2', \dots, g_L')$,
 - unde $g_i^1, g_i^2, g_i' \in [LI_i, LS_i]$, pt. $i=1, 2, \dots, L$
- Încrucișare amestecată (*blend crossover – BLX*)
 - Ideea de bază
 - Se generează un singur descendent
 - Genele g_i' ale descendentului sunt alese aleator în intervalul $[Min_i - I * a, Max_i + I * a]$, unde:
 - $Min_i = \min\{g_i^1, g_i^2\}$, $Max_i = \max\{g_i^1, g_i^2\}$
 - $I = Max - Min$, a – parametru din $[0, 1]$

$[LI, LS] = [-2.5, +3]$

$a = 0.7$

0.3	1.5	3.2	2.4	1.1	0.6	2.0	1.7
2.1	1.3	0.2	1.4	1.1	0.3	1.0	1.7



1.25	1.45	-1.11	2.37	1.10	0.11	0.70	1.70
------	------	-------	------	------	------	------	------

Min	0.3	1.3	0.2	1.4	1.1	0.3	1.0	1.7
Max	2.1	1.5	3.2	2.4	1.1	0.6	2.0	1.7
I	0.8	0.2	3.0	1.0	0	0.3	1.0	0.0

Min-Ia	-0.26	1.16	-1.90	0.70	1.10	0.09	0.30	1.70
Max+Ia	2.66	1.50	3.20	2.40	1.10	0.60	2.00	1.70

Algoritmi evolutivi – algoritm

Proiectare – recombinaarea (repmez. reală)



- Din 2 cromozomi părinți
 - $p_1 = (g_1^1, g_2^1, \dots, g_L^1)$ și $p_2 = (g_1^2, g_2^2, \dots, g_L^2)$
- se obțin 2 descendenți
 - $c_1 = (g_1', g_2', \dots, g_L')$ și $c_2 = (g_1'', g_2'', \dots, g_L'')$,
 - unde $g_i^1, g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, pt. $i=1, 2, \dots, L$
- Încrucișare binară simulată
 - Ideea de bază
 - Fiecare genă a unui descendent reprezintă o combinație a genelor părinților
$$d_1 = \frac{p_1 + p_2}{2} - \beta \frac{p_2 - p_1}{2}, \quad d_2 = \frac{p_1 + p_2}{2} + \beta \frac{p_2 - p_1}{2}$$
 - a.î. să se respecte cele 2 proprietăți de la încrucișarea cu n puncte de tăietură (pt. reprezentarea binară)
 - media valorilor codate în părinți = media valorilor codate în descendenți
 - probabilitatea apariției unui factor de răspândire $\beta \approx 1$ este mai mare decât a oricărui alt factor

Algoritmi evolutivi – algoritm

Proiectare – recombinaarea



□ Recombinarea multiplă

- Bazată pe frecvența valorilor din părinți (încrucișare uniformă generală)
- Bazată pe segmentare și recombinaare (încrucișare generală cu puncte de tăietură diagonală)
- Bazată pe operații numerice specifice valorilor reale (încrucișare bazată pe centrul de masă, încrucișare generală aritmetică)

Algoritmi evolutivi – algoritm

Proiectare – mutație sau recombinare?

- Dezbateri aprinse
 - Întrebări:
 - care operator este mai bun?
 - care operator este necesar?,
 - care operator este mai important?
 - Răspunsuri:
 - Depinde de problemă, dar
 - În general, este bine să fie folosiți ambii operatori
 - Fiecare având alt rol
 - Sunt posibili AE doar cu mutație, dar nu sunt posibili AE doar cu încrucișare
- Aspecte ale căutării:
 - Explorare → descoperirea regiunilor promițătoare ale spațiului de căutare (acumulând informație utilă despre problemă)
 - Exploatare → optimizarea într-o regiune promițătoare (folosind informația existentă)
 - Trebuie să existe cooperare și competiție între aceste 2 aspecte
- Încrucișarea
 - Operator exploativ, realizând un mare salt într-o regiune undeva între regiunile asociate părinților
 - Efectele exploative se reduc pe măsură ce AE converge
 - Operator binar (n-ar) care poate combina informația din 2 (sau mai mulți) părinți
 - Operator care nu schimbă frecvența valorilor din cromozomi la nivelul întregii populații
- Mutația
 - Operator explorativ, realizând mici diversiuni aleatoare, rămânând în regiunea apropiată părintelui
 - Evadarea din optimele locale
 - Operator care poate introduce informație genetică nouă
 - Operator care schimbă frecvența valorilor din cromozomi la nivelul întregii populații

Algoritmi evolutivi – algoritm

Proiectare – criteriu de oprire



- Stabilirea unui criteriu de stop
 - S-a identificat soluția optimă
 - S-au epuizat resursele fizice
 - S-a efectuat un anumit număr de evaluaări ale funcției de fitness
 - S-au epuizat resursele utilizatorului (timp, răbdare)
 - S-au "născut" câteva generații fără îmbunătățiri

Algoritmi evolutivi - algoritm



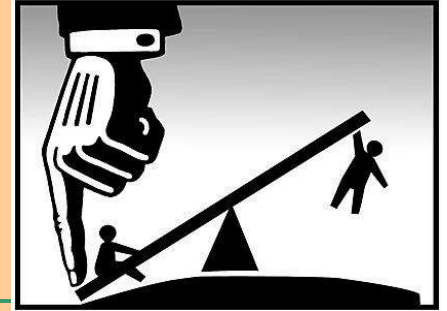
- Evaluarea performanțelor unui AE
 - După mai multe rulări se calculează:
 - Măsuri statistice
 - media soluțiilor,
 - mediana soluțiilor,
 - cea mai bună soluție,
 - cea mai slabă soluție,
 - deviația standard – pentru comparabilitate
 - Calculate pentru un număr suficient de mare de rulări independente

Algoritmi evolutivi



- Analiza complexității
 - Partea cea mai costisitoare → calculul fitnessului

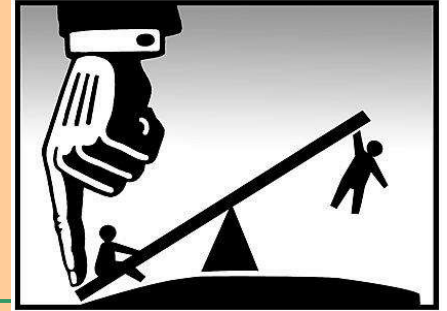
Algoritmi evolutivi



□ Avantaje

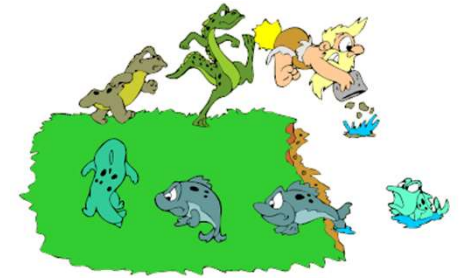
- Schema AE universală pentru toate problemele
 - se modifică doar
 - reprezentarea
 - funcția de fitness
- AE sunt capabili să producă rezultate mai bune decât metodele convenționale de optimizare pentru că:
 - nu necesită liniarizare
 - nu implică anumite presupuneri (continuitate, derivabilitate, etc. a funcției obiectiv)
 - nu ignoră anumite potențiale soluții
- AE sunt capabili să exploreze mai multe potențiale soluții decât poate explora omul

Algoritmi evolutivi



- Dezavantaje
 - Timp de rulare îndelungat

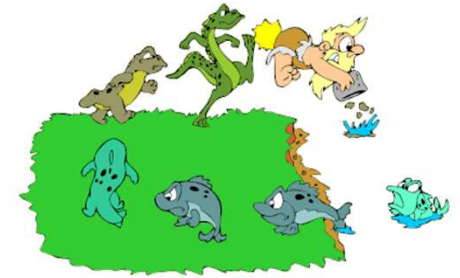
Algoritmi evolutivi



□ Aplicații

- Proiectări vehicule
 - Componenta materialelor
 - Forma vehiculelor
- Proiectări ingineresti
 - Optimizarea structurală și organizatorică a construcțiilor (clădiri, roboți, sateliți, turbine)
- Robotică
 - Optimizarea proiectării, funcționării componentelor
- Evoluare de hardware
 - Optimizarea de circuite digitale
- Optimizarea telecomunicațiilor
- Generarea de glume și jocuri de cuvinte
- Invenții biomimetice (inspirate de arhitecturi naturale)
- Rutări pentru trafic și transporturi
- Jocuri de calculator
- Criptări
- Profilul expresiv al genelor
- Analiza chimică a cinecticii
- Strategii financiare și marketing

Algoritmi evolutivi



- Tipuri de algoritmi evolutivi
 - Strategii evolutive
 - Programare evolutivă
 - Algoritmi genetici
 - Programare genetică

Cursul următor

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

Cursul următor – Materiale de citit și legături utile

- ❑ capitolul 16 din *C. Groșan, A. Abraham, Intelligent Systems: A Modern Approach, Springer, 2011*
- ❑ *James Kennedy, Russel Eberhart, Particle Swarm Optimisation, Proceedings of IEEE International Conference on Neural Networks. **IV**. pp. 1942–1948, 1995*
(04_ACO_PSO/PSO_00.pdf)
- ❑ *Marco Dorigo, Christian Blum, Ant colony optimization theory: A survey, Theoretical Computer Science 344 (2005) 243 – 27* (04_ACO_PSO/Dorigo05_ACO.pdf)

-
- 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
 - Lect. Dr. Crina Groșan - www.cs.ubbcluj.ro/~cgrosan
 - Prof. Dr. Horia F. Pop - www.cs.ubbcluj.ro/~hfpop