UNIVERSITATEA BABEŞ-BOLYAI
Facultatea de Matematică şi Informatică

INTELIGENȚĂ ARTIFICIALĂ

Rezolvarea problemelor de căutare

Strategii de căutare informată locală

Algoritmi Evolutivi

Laura Dioşan

Sumar

A. Scurtă introducere în Inteligenţa 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 învaţă singure
  - Arbori de decizie
  - Reţele neuronale artificiale
  - Maşini 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 şi legături utile

- capitolul 14 din *C. Groşan, 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

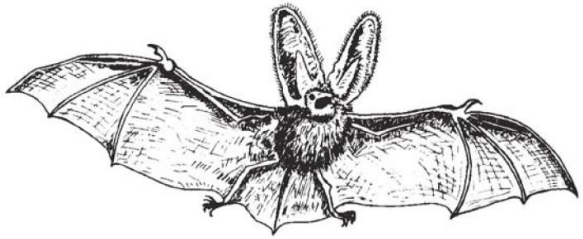
- Tipologie
  - Căutare locală simplă - se reţine o singură stare vecină
    - Hill climbing → alege cel mai bun vecin
    - Simulated annealing → alege probabilistic cel mai bun vecin
    - Căutare tabu → reţine lista soluţiilor recent vizitate
  - Căutare locală în fascicol (beam local search) – se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

Algoritmi inspirați de natură

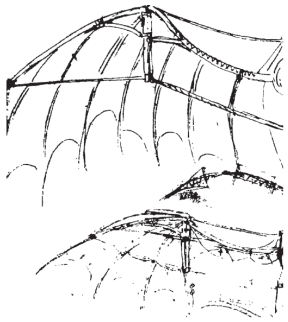
- Care este cea mai bună metodă de rezolvare a unei probleme?
  - Creierul uman
    - a creat roata, maşina, oraşul, etc
  - Mecanismul evoluţiei
    - a creat creierul (mintea) umană
- Simularea naturii
  - Cu ajutorul maşinilor → reţelele neuronale artificiale simulează mintea umană
    - maşini de zbor, computere bazate pe ADN, computere cu membrane
  - Cu ajutorul algoritmilor
    - algoritmii evolutivi simulează evoluţia naturii
    - algoritmii inspiraţi de comportamentul de grup simulează adaptarea colectivă şi procesele sociale dintr-un colectiv (*Particle Swarm Optimisation*)
    - algoritmii inspiraţi de furnici (*Ant Colony Optimisation*)

Algoritmi evolutivi – aspecte teoretice

- Simularea naturii
  - Zborul liliecilor

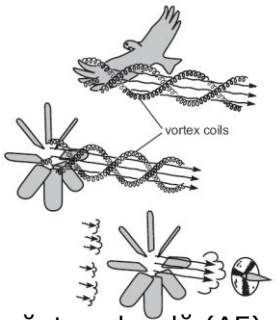


  - Leonardo da Vinci – schema unei maşini de zbor



  - Zborul păsărilor şi al avioanelor

  - Zborul păsărilor şi turbinele eoliene



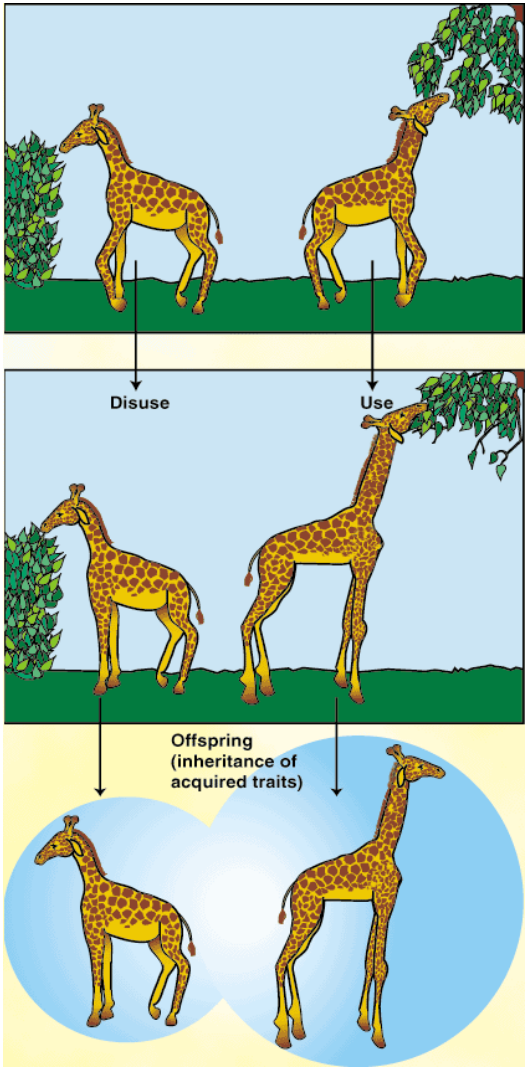
Algoritmi evolutivi – aspecte teoretice

- Care sunt caracteristicile de bază ale AE?
  - Implică procese iterative şi paralele
  - Folosesc populaţii de potenţiale soluţii
  - Se bazează pe o căutare aleatoare
  - Sunt inspiraţi de biologie – implică mecanisme precum:
    - selecţia naturală
    - reproducerea
    - recombinarea
    - mutaţia

Algoritmi evolutivi – aspecte teoretice

Câteva repere istorice

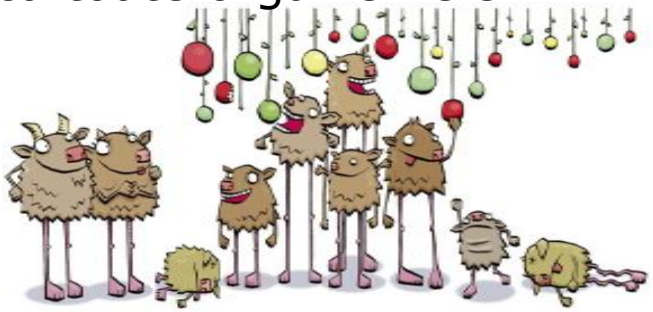
- Jean Baptise de Lamark (1744-1829)
  - A propus în 1809 o explicaţie pentru originea speciilor în cartea *Zoological Philosophy*:
    - Nevoile unui organism determină caracteristicile care evoluează
    - Caracteristicile utile dobândite în cursul vieţii unui organism se pot transfera urmaşilor acestuia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

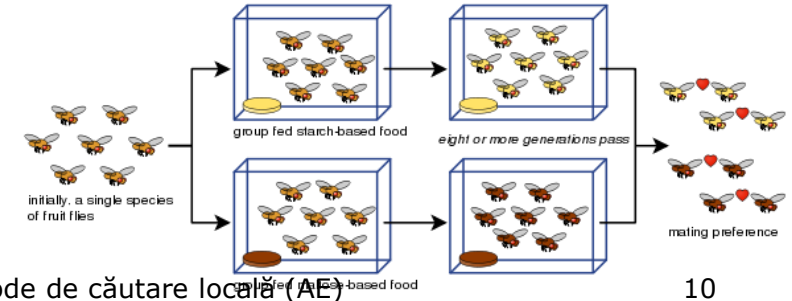


Algoritmi evolutivi – aspecte teoretice

Câteva repere istorice

- Charles Darwin (1807-1882)
  - În cartea *Origin of Species* demostrează că toate organismele au evoluat din alte organisme pe baza:
    - variaţiei
      - supraproducţia de descendenţi
    - selecţiei naturale
      - competiţia (generaţii constante ca dimensiune)
      - supravieţuirea pe baza calităţii/adaptării la mediul de viaţă (fitness)
      - reproducerea
      - apariţia de specii noi





Algoritmi evolutivi – aspecte teoretice

Câteva repere istorice

- Teoria evolutivă modernă
  - Îmbogăţeşte teoria Darwiniană cu mecanismul moştenirii genetice
  - Variaţia genetică se produce prin:
    - mutaţie – spontană – şi
    - 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

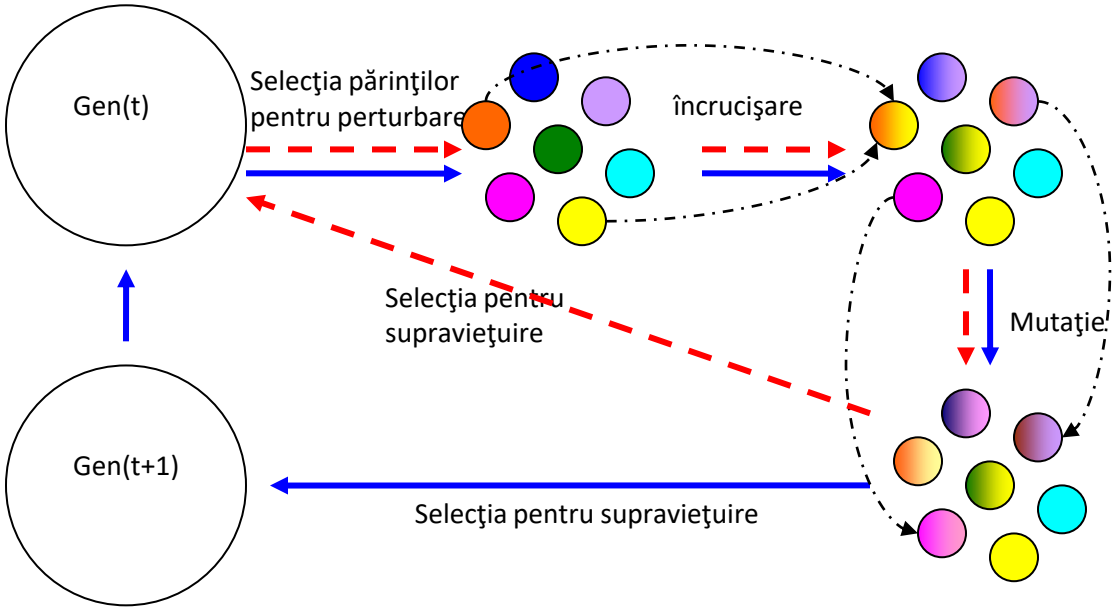
Evoluţia naturală		Rezolvarea problemelor
Individ	↔	Soluţie potenţială
Populaţie	↔	Mulţime de soluţii potenţiale
Cromozom	↔	Codarea unei soluţii potenţiale
Genă	↔	Parte a codării
Fitness	↔	Calitate
Încrucişare şi mutaţie	↔	Operatori de căutare
Mediu	↔	Problemă

Algoritmi evolutivi - algoritm

- Schema generală
- Proiectare

Algoritmi evolutivi – algoritm

- Schema generală a unui AE
  - Generaţional ——→
  - Steady-state - - - - →



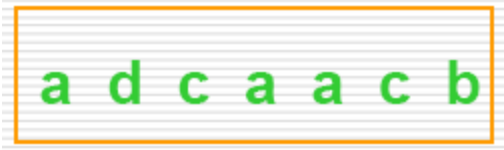
Algoritmi evolutivi – algoritm

- Proiectare
  - Alegerea unei reprezentări a cromozomilor
  - Alegerea unui model de populaţie
  - Stabilirea unei funcţii de evaluare
  - Stabilirea operatorilor genetici
    - Selecţie
    - Mutaţie
    - Recombinare
  - Stabilirea unui criteriu de stop

Algoritmi evolutivi – algoritm
Proiectare – alegerea unei reprezentări

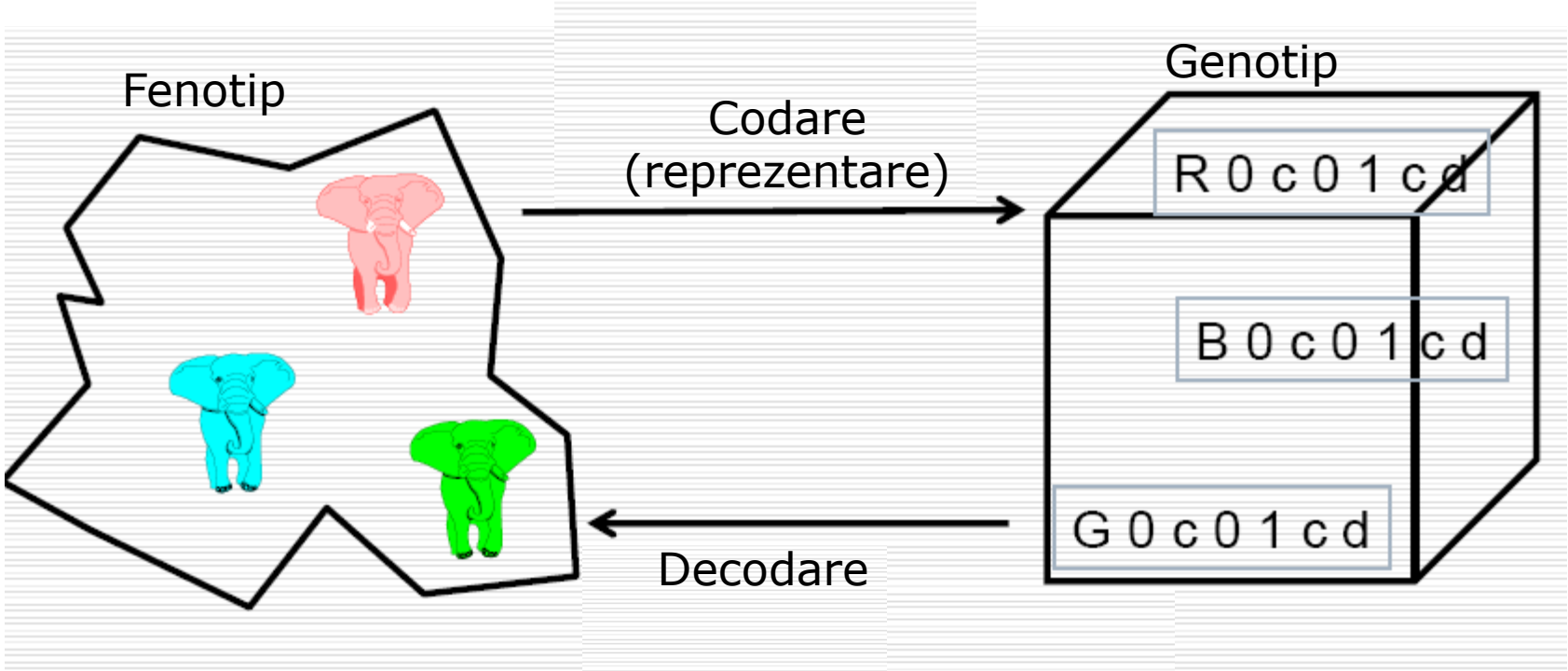
- 2 nivele de existenţă pentru o soluţie candidat
  - Nivel exterior → fenotip
    - Individ - obiectul original în contextul dat de problemă
    - Aici are loc evaluarea unei potenţiale soluţii
    - Furnică, rucsac, elefant, oraşe, ...

  - Nivel interior → genotip
    - Cromozom – codul asociat unui obiect
      - format din gene, poziţionate în locuri (fixe) – loci – şi având anumite valori – alele
    - Aici are loc căutarea unei noi potenţiale soluţii
    - Vector unidimensional (numeric, boolean, string), matrice, ...



Algoritmi evolutivi – algoritm
Proiectare – alegerea unei reprezentări

- Reprezentarea trebuie să fie relevantă pentru:
  - problemă,
  - funcţia de evaluare şi
  - operatorii genetici



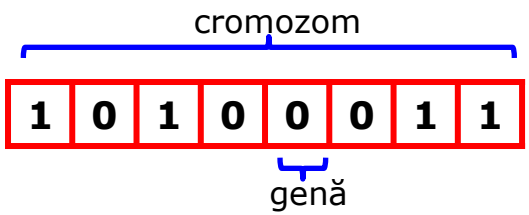
Algoritmi evolutivi – algoritm
Proiectare – alegerea unei reprezentări

Tipologia reprezentării cromozomilor

- Liniară
  - Discretă
    - Binară → problema rucsacului
    - Ne-binară
      - Întreagă
        - Oarecare → procesarea imaginilor
        - Permutări → problema comisului voiajor
      - Categorială → problema colorării hărţilor
  - Continuă (reală) → optimizări de funcţii
- Arborescentă → probleme de regresie

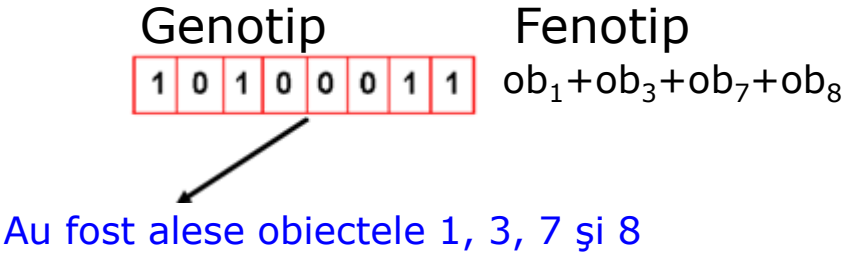
Algoritmi evolutivi – algoritm
Proiectare – alegerea unei reprezentări

- Reprezentare liniară discretă binară
  - Genotip
    - şir de biţi



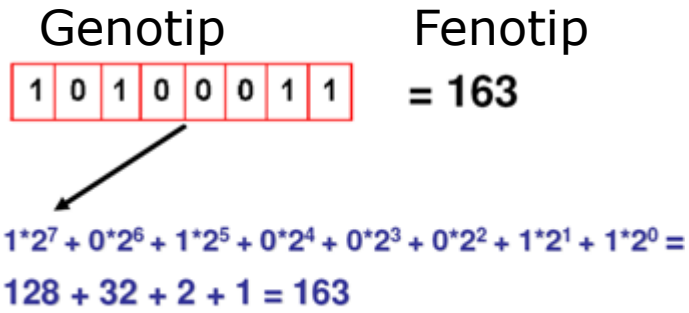
Algoritmi evolutivi – algoritm
Proiectare – alegerea unei reprezentări

- Reprezentare liniară discretă binară
  - Genotip
    - şir de biţi
  - Fenotip
    - Elemente de tip Boolean
      - Ex. Problema rucsacului – obiectele alese pentru umplerea rucsacului



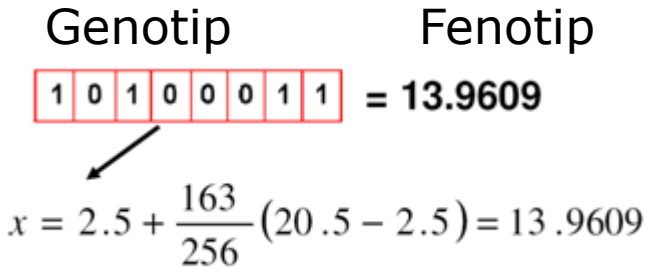
Algoritmi evolutivi – algoritm
Proiectare – alegerea unei reprezentări

- Reprezentare liniară discretă binară
  - Genotip
    - şir de biţi
  - Fenotip
    - Elemente de tip Boolean
      - Ex. Problema rucsacului – obiectele alese pentru umplerea rucsacului
    - Numere întregi



Algoritmi evolutivi – algoritm
Proiectare – alegerea unei reprezentări

- Reprezentare liniară discretă binară
  - Genotip
    - şir de biţi
  - 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])



Algoritmi evolutivi – algoritm
Proiectare – alegerea unei reprezentări

Transformarea valorilor reale reprezentate pe biţi

- Fie $z \in [x,y] \subseteq \mathcal{R}$ reprezentat ca $\{a_1,\ldots,a_L\} \in \{0,1\}^L$
- Funcţia $[x,y] \rightarrow \{0,1\}^L$ trebuie să fie inversabilă (un fenotip corespunde unui genotip)
- Funcţia Γ: $\{0,1\}^L \rightarrow [x,y]$ defineşte reprezentarea

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

- Observaţii
  - Se pot reprezenta doar 2^L valori
  - L indică precizia maximă a soluţiei
  - Pentru o precizie cât mai bună → cromozomi lungi ➔ evoluţie încetinită

Algoritmi evolutivi – algoritm
Proiectare – alegerea unei reprezentări

- Reprezentare liniară discretă ne-binară întreagă oarecare
  - Genotip
    - şir de numere întregi dintr-un anumit interval
  - Fenotip
    - Utilitatea numerelor în problemă
  - Ex. Problema plăţii unei sume folosind diferite monezi
    - Genotip → şir de nr întregi de lungime egală cu numărul de monezi diferite, fiecare număr din intervalul [0,suma/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 oraşelor, ştiind că fiecărui oraş îi corespunde un număr din mulţimea {1,2,...,*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
    - şir de etichete dintr-o anumită mulţime
  - Fenotip
    - Interpretarea etichetelor
  - Ex. Problema colorării hărţilor
    - Genotip → şir de etichete (culori) de lungime egală cu numărul de ţări, fiecare etichetă aparţinând unei mulţimi de culori date
    - Fenotip → cu ce culoare trebuie haşurată fiecare hartă a unei ţări

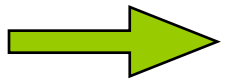
Algoritmi evolutivi – algoritm
Proiectare – alegerea unei reprezentări

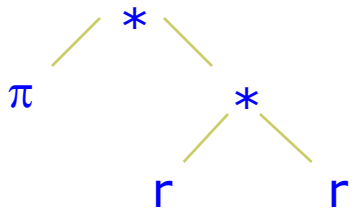
- Reprezentare liniară continuă (reală)
  - Genotip
    - Şir de numere reale
  - Fenotip
    - Utilitatea numerelor în problemă
  - Ex. Problema optimizării funcţiilor $f: R^n \rightarrow R$
    - Genotip → tuplu de numere reale $X=[x_1, x_2, ..., x_n]$, $x_i \in R$
    - Fenotip → valorile asociate argumentelor funcţiei *f*

Algoritmi evolutivi – algoritm
Proiectare – alegerea unei reprezentări

- Reprezentare arborescentă
  - Genotip
    - Arbori care codează S-Expresii
    - Nodurile interne ale arborelui → funcţii (F)
      - Matematice
        - Operatori aritmetici
        - Operatori de tip Boolean
      - Instrucţiuni
        - Într-un limbai de programare
        - Alt tip de instrucţiuni
    - Frunzele arborelui → terminale (T)
      - Valori reale sau Booleene, constante sau variabile
      - Subprograme
  - Fenotip
    - Interpretarea S-expresiilor
  - Ex. Calculul ariei unui cerc

$$\pi * r^2$$





Algoritmi evolutivi – algoritm
Proiectare – formarea unei populații

- Populaţie – concept
  - Scop
    - reţine o colecţie de soluţii candidat
      - se permit repetiţii
    - este folosită în întregime în procesul de selecţie pentru reproducere
  - Proprietăţi
    - dimensiune (de obicei) fixă μ
    - diversitate
      - Nr de fitness-uri/fenotipuri/genotipuri diferite
  - Observaţii
    - Reprezintă unitatea de bază care evoluează
      - populaţia întreagă evoluează, nu indivizii!!!

Algoritmi evolutivi – algoritm
Proiectare – formarea unei populații

- Populaţie – iniţializare
  - Uniformă (dacă e posibil) în spaţiul de căutare
    - Stringuri binare
      - generarea de 0 şi 1 cu probabilitatea 0.5
    - Şiruri de numere reale generate uniform (într-un anumit interval)
    - Permutări
      - generarea permutării identice şi efectuarea unor schimbări

Algoritmi evolutivi – algoritm
Proiectare – formarea unei populații

- Populaţie – iniţializare
  - Uniformă (dacă e posibil) în spaţiul de căutare
    - Arbori
      - Metoda *Full* – arbori compleţi
        - Nodurile de la adâncimea d < D_{max} se iniţializează aleator cu o funcţie din setul de funcţii *F*
        - Nodurile de la adâncimea d = D_{max} se iniţializează aleator cu un terminal din setul de terminale *T*
      - Metoda *Grow* – arbori incompleţi
        - Nodurile de la adâncimea d < D_{max} se iniţializează aleator cu un element din *F U T*
        - Nodurile de la adâncimea d = D_{max} se iniţializează aleator cu un terminal din setul de terminale *T*
      - Metoda *Ramped half and half*
        - ½ din populaţie se creează cu metoda *Full*
        - ½ din populaţie se creează cu metoda *Grow*
        - Folosind diferite adâncimi

Algoritmi evolutivi – algoritm
Proiectare – formarea unei populații

- Modele de populaţii – algoritm evolutiv:
  - Generaţional
    - În fiecare generaţie se crează μ descendenţi
    - Fiecare individ supravieţuieşte o singură generaţie
    - Mulţimea părinţilor este înlocuită în întregime cu mulţimea descendenţilor
  - Steady-state
    - În fiecare generaţie se obţine un singur descendent
    - Un singur părinte (cel mai slab) este înlocuit cu descendentul obţinut

- Discrepanţa între generaţii (*Generation Gap*)
  - Proporţia populaţiei înlocuite
  - 1 = μ/μ, pentru modelul generaţional
  - $1/\mu$, pentru modelul steady-state

Algoritmi evolutivi – algoritm
Proiectare – funcția de evaluare

- Scop
  - Reflectă condiţiile la care trebuie să se adapteze populaţia
  - Funcţie de calitate sau funcţie obiectiv
  - Asociază o valoare fiecărei soluţii candidat
    - Consecinţe asupra selecţiei → cu cât sunt mai multe valori diferite, cu atât e mai bine

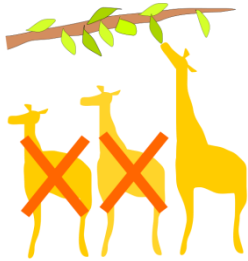
- Proprietăţi
  - Etapa cea mai costisitoare
    - Nu se re-evaluează indivizii nemodificaţi

- Tipologie:
  - După nr de obiective urmărite:
    - Uni-obiectiv
    - Multi-obiectiv → fronturi Pareto
  - După direcţia 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 → abs(greutatea rucsacului – greutatea obiectelor alese) → minimizare
  - Problema plăţii unei sume folosind diferite monezi
    - reprezentare → liniară discretă întreagă
    - fitness → abs(suma de plată – suma monezilor selectate) → minimizare
  - Problema comisului voiaior
    - reprezentare → liniară discretă întreagă sub formă de permutare
    - fitness → costul drumului parcurs → minimizare
  - Problema optimizării funcţiilor
    - Reprezentare → liniară continuă reală
    - fitness → valoarea funcţiei → minimizare/maximizare
  - Calculul ariei unui cerc
    - reprezentare → arborescentă
    - fitness → suma pătratelor erorilor (diferenţelor între valoarea reală şi cea calculată pe un set de exemple) → minimizare

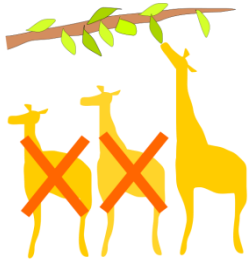
Algoritmi evolutivi – algoritm
Proiectare – selecția



- Scop:
  - acordă şanse de reproducere/supravieţuire mai mari indivizilor mai buni
    - şi indivizii mai slabi trebuie să aibă şansa să se reproducă/supravieţuiască pentru că pot conţine material genetic util
  - direcţionează populaţia spre îmbunătăţirea calităţii

- Proprietăţi
  - lucrează la nivel de populaţie
  - se bazează doar pe fitnessul indivizilor (este independentă de reprezentare)
  - aiută la evadarea din optimele locale datorită naturii sale stocastice

Algoritmi evolutivi – algoritm
Proiectare – selecția

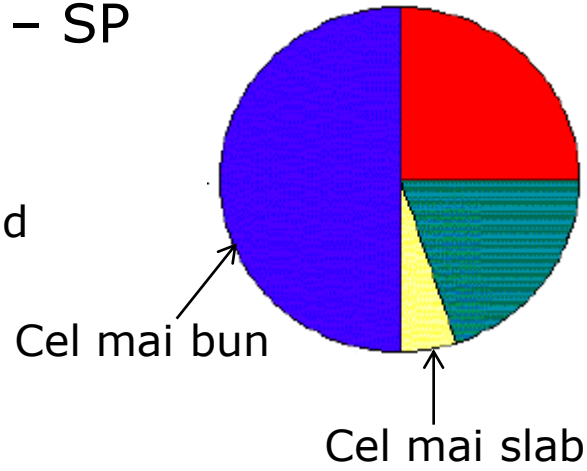


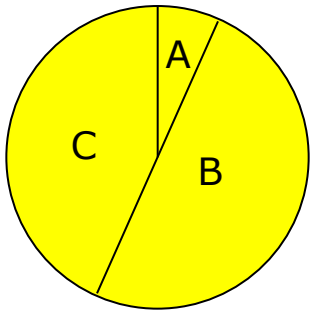
- Tipologie
  - în funcţie de scop:
    - Selecţia părinţilor (din generaţia curentă) pentru reproducere
    - Selecţia supravieţuitorilor (din părinţi şi descendenţi) pentru generaţia următoare
  - în funcţie de modul de decidere al câştigătorului
    - Deterministă – cel mai bun câştigă
    - Stocastică – cel mai bun are cele mai mari şanse să câştige
  - în funcţie de mecanism
    - Selecţia pentru reproducere
      - Selecţie proporţională (bazată pe fitness)
      - Selecţie bazată pe ranguri

      } Bazate pe întreaga populaţie
      - Selecţie prin turnir ----→ Bazată pe o parte din populaţie
    - Selecţia pentru supravieţuire
      - Bazată pe vârstă
      - Bazată pe calitate (fitness)

Algoritmi evolutivi – algoritm
Proiectare – selecția pt. reproducere

- Selecţie proporţională (bazată pe fitness) – SP
  - Ideea de bază
    - Algoritmul ruletei la nivelul întregii populaţii
    - Estimarea numărului de copii ale unui individ $E(n_i) = \mu \frac{f(i)}{\langle f \rangle}$, unde:
      - μ = dimensiunea populaţiei,
      - *f(i)* = fitnessul individului *i*,
      - ⟨f⟩ = fitnessul mediu al populaţiei
    - Indivizii mai buni
      - au alocat mai mult spaţiu în ruletă
      - au şanse mai mari să fie selectaţi
    - Ex. O populaţie cu $\mu = 3$ indivizi



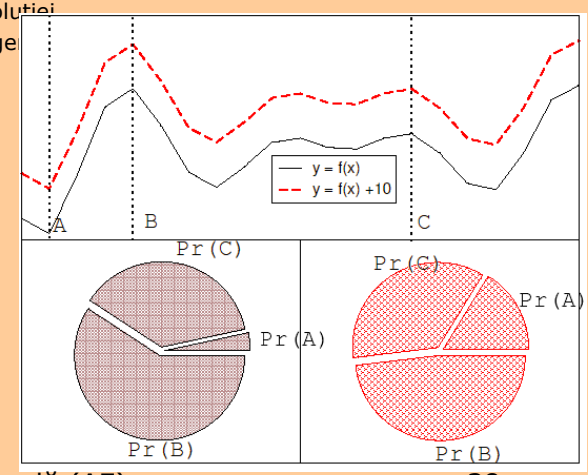


	f(i)	$P_{selSP}(i)$
A	1	1/10=0.1
B	5	5/10=0.5
C	4	4/10=0.4
Suma	10	1

Algoritmi evolutivi – algoritm
Proiectare – selecția pt. reproducere

Selecţie proporţională (bazată pe fitness)

- Avantaie
  - Algoritm simplu
- Dezavantaie
  - Convergenţa prematură
    - cromozomii foarte buni tind să domine populaţia
  - Presiune de selecţie foarte mică atunci când fintessurile indivizilor sunt foarte apropiate (la sfârşitul rulării)
  - Susceptibilă de traspoziţia funcţiei
  - Rezultatele reale ale unei astfel de selcţii diferă de distribuţia probabilistică teoretică
  - Lucrează cu întreaga populaţie
- Soluţii
  - 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 populaţia curentă (a *t*-a ge
    - 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 populaţei
        - σ_f – deviaţia standard a fitnessului populaţiei
    - Scalare prin normalizare
      - Se începe cu fitnessurile absolute (iniţiale)
      - Se standardizează astfel încât Se aiustează fitnessurile a.î.:
        - ele să aparţină [0,1]
        - cel mai bun fitness să fie cel mai mic (egal cu 0)
        - suma lor să fie 1
  - alt mecanism de selecţie



Algoritmi evolutivi – algoritm
Proiectare – selecția pt. reproducere

- Selecţia bazată pe ranguri – SR
  - Ideea de bază
    - Se ordonează întreaga populaţie pe baza fitnessului
      - Creşte puţin complexitatea algoritmului, dar se poate negliia această creştere comparativ cu timpul necesar evaluării unui individ
    - Se acordă ranguri fiecărui individ
    - Se calculează probabilităţile de selecţie pe baza rangurilor
      - Cel mai slab individ are rangul 1
      - Cel mai bun individ are rangul μ
    - Încearcă să rezolve problemele selecţiei proporţionale prin folosirea fitnessurilor relative (în locul celor absolute)

Algoritmi evolutivi – algoritm
Proiectare – selecția pt. reproducere

- Selecţia bazată pe ranguri – SR
  - Modalităţi de acordare a rangurilor
    - Liniară (RL) $P_{lin\_rank}(i) = \frac{2-s}{\mu} + \frac{2i(s-1)}{\mu(\mu-1)}$
      - *s* – presiunea de selecţie
        - măsoară avantaiele celui mai bun individ
        - $1.0 < s \leq 2.0$
        - în algoritmul genetic generaţional *s* este numărul de copii ai unui individ
      - Ex. pentru o populaţie cu μ = 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			

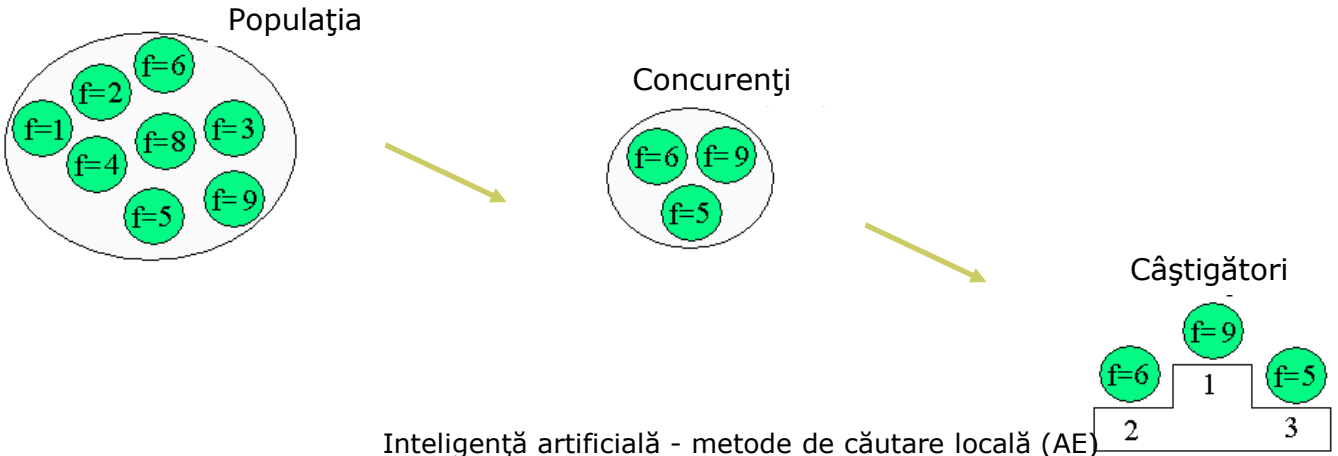
    - Exponenţială (RE) $P_{\exp\_rank}(i) = \frac{1-e^{-i}}{c}$
      - Cel mai bun individ poate avea mai mult de 2 copii
      - c – factor de normalizare
        - depinde de dimensiunea populaţiei (μ)
        - trebuie ales a.î. suma probabilităţilor de selecţie să fie 1

Algoritmi evolutivi – algoritm
Proiectare – selecția pt. reproducere

- Selecţia bazată pe ranguri – SR
  - Avantaie
    - Păstrează presiunea de selecţie constantă
  - Dezavantaie
    - Lucrează cu întreaga populaţie
  - Soluţii
    - Alt mecanism de selecţie

Algoritmi evolutivi – algoritm
Proiectare – selecția pt. reproducere

- Selecţia prin turnir
  - Ideea de bază
    - Se aleg aleator *k* indivizi → eşantion de *k* indivizi (*k* – mărimea turnirului)
    - Se selectează cel mai bun individ dintre cei aleşi anterior
    - Probabilitatea alegerii unui individ în eşantion depinde de
      - Rangul individului
      - Dimensiunea eşantionului (*k*)
        - Cu cât k este mai mare, cu atât creşte şi presiunea de selcţie
      - Modul în care se face alegerea – dacă se realizează cu înlocuire (model steady-state) sau nu
        - Alegerea fără înlocuire creşte presiunea de selecţie
      - Pt k = 2 timpul necesar ca cel mai bun individ să domine populaţia este acelaşi cu cel de la selecţia pe bază de ranguri liniare cu s = 2 * *p*, *p* – probabilitatea alegerii celui mai bun individ din populaţie



Algoritmi evolutivi – algoritm
Proiectare – selecția pt. reproducere

- Selecţia prin turnir

  - Avantaje
    - Nu implică lucrul cu întrega populaţie
    - Uşor de implementant
    - Uşor de controlat presiunea de selcţie prin intermediul parametrului k

  - Dezavantaje
    - Rezultatele reale ale unei astfel de selecţii diferă de distribuţia probabilistică teoretică (similar selecţiei prin mecanismul ruletei)

Algoritmi evolutivi
Proiectare – selecția

- Selecţia pentru supravieţuire (înlocuire)
  - Pe baza vârstei
    - eliminarea celor mai "bătrâni" indivizi
  - Pe baza calităţii (fitness-ului)
    - selecţiei proporţională
    - selecţie bazată pe ranguri
    - selecţie prin turnir
    - elitism
      - Păstrarea celor mai buni indivizi de la o generaţie la alta (dacă descendenţii sunt mai slabi ca părinţii se păstrează părinţii)
    - GENITOR (înlocuirea celui mai slab individ)
      - Eliminarea celor mai slabi λ indivizi

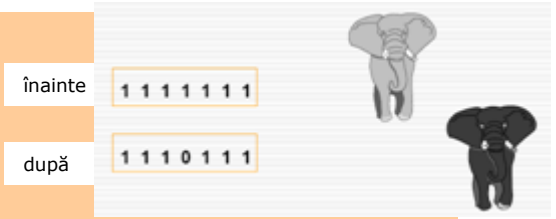
Algoritmi evolutivi - algoritm
Proiectare – operatori de variație

- Scop:
  - Generarea unor soluţii potenţiale noi
- Proprietăţi
  - lucrează la nivel de individ
  - se bazează doar pe reprezentarea indivizilor (independent de fitness)
  - Aiută la explorarea şi exploatarea spaţiului de căutare
  - Trebuie să producă indivizi valizi
- Tipologie
  - În funcţie de aritate
    - Aritate 1 → operatori de mutaţie
    - Aritate > 1 → operatori de recombinare/încrucişare

Algoritmi evolutivi – algoritm
Proiectare – mutația

- Scop
  - Reintroducerea în populaţie a materialului genetic pierdut
  - Operator unar de căutare (spaţiul continuu)
  - Introducerea diversităţii în populaţie (în spaţiul discret – binar)

- Proprietăţi
  - Acţionează la nivel de genotip
  - Bazată pe elemente aleatoare



  - Responsabilă cu explorarea unor noi regiuni promiţătoare ale spaţiului de căutare
  - Este responsabilă de evadarea din optimele locale

  - Trebuie să producă mici schimbări stocastice ale individului
  - Mărimea mutaţiei 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
    - Mutaţie tare - bit-flipping
    - Mutaţie slabă
  - Reprezentare întreagă
    - Random resetting
    - Creep mutation
  - Reprezentare permutare
    - Mutaţie prin inserţie
    - Mutaţie prin interchimbare
    - Mutaţie prin inversare
    - Mutaţie prin amestec
    - Mutaţie k-opt
  - Reprezentare reală
    - Mutaţie uniformă
    - Mutaţie neuniformă
      - Mutaţie Gaussiană
      - Mutaţie Cauchy
      - Mutaţie Laplace
  - Reprezentare arborescentă → într-un curs viitor
    - Mutaţie grow
    - mutaţie shrink
    - Mutaţie switch
    - Mutaţie cycle
    - Mutaţie tip Koza
    - Mutaţie pentru terminalele numerice

Algoritmi evolutivi – algoritm
Proiectare – mutația (reprez. binară)

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

- Mutaţie tare – *bit flipping*
  - Ideea de bază
    - Schimbarea cu probabilitatea p_m (rată de mutaţie) a unor gene în complementul lor
      - 1 → 0
      - 0 → 1
    - Ex. Un cromozom cu L = 8 gene, p_m = 0.1

Algoritmi evolutivi – algoritm
Proiectare – mutația (reprez. binară)

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

- Mutaţie slabă
  - Ideea de bază
    - Schimbarea cu probabilitatea p_m (rată de mutaţie) a unor gene în 0 sau 1
      - 1 → 0/1
      - 0 → 1/0
    - Ex. Un cromozom cu L = 8 gene, p_m = 0.1

Algoritmi evolutivi – algoritm
Proiectare – mutația (reprez. întreagă)

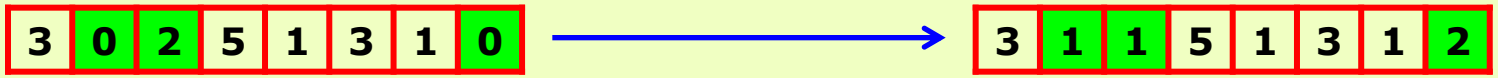
- Un cromozom $c=(g_1,g_2,...,g_L)$ devine $c'=(g_1',g_2',...,g_L')$, unde $g_i, g_i' \in \{val_1, val_2,...,val_k\}$, pt. $i=1,2,...,L$

- Mutaţie *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 $c=(g_1,g_2,...,g_L)$ devine $c'=(g_1',g_2',...,g_L')$, unde $g_i, g_i' \in \{val_1, val_2, ..., val_k\}$, pt. $i=1,2,...,L$

- Mutaţie *creep*
  - Ideea de bază
    - Valoarea unei gene este schimbată (cu probabilitatea p_m) prin adăugarea unei valori (pozitivă sau negativă)
      - valoarea → face parte dintr-o distribuţie simetrică faţă de zero
      - modificarea produsă este fină (mică)



Algoritmi evolutivi – algoritm
Proiectare – mutația (reprez. permutare)

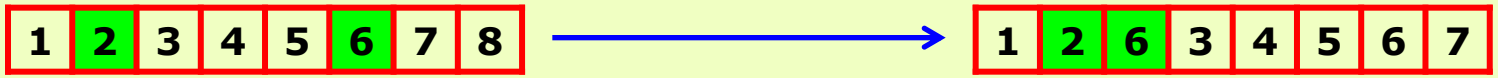
- Un cromozom $c=(g_1,g_2,...,g_L)$ cu $g_i \neq g_i$ pentru orice $i \neq i$ devine $c'=(g_1',g_2',...,g_L')$, unde $g_i, g_i' \in \{val_1, val_2,...,val_L\}$, pt. $i=1,2,...,L$ a.î. $g_i' \neq g_i'$ pentru orice $i \neq i$.

- Mutaţie prin interschimbare (*swap mutation*)
  - Ideea de bază
    - Se aleg aleator 2 gene şi se interschimbă valorile lor

Algoritmi evolutivi – algoritm
Proiectare – mutația (reprez. permutare)

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

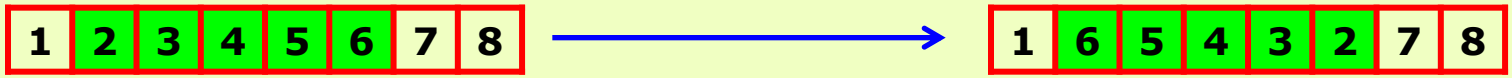
- Mutaţie prin inserţie
  - Ideea de bază
    - Se aleg 2 gene oarecare g_i şi g_j cu $j > i$
    - Se inserează gj după gi a.î. $g_i'=g_i$, $g_{i+1}'=g_j$, $g_{k+2}'=g_{k+1}$, pentru k=i, i+1, i+2, ...



Algoritmi evolutivi – algoritm
Proiectare – mutația (reprez. permutare)

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

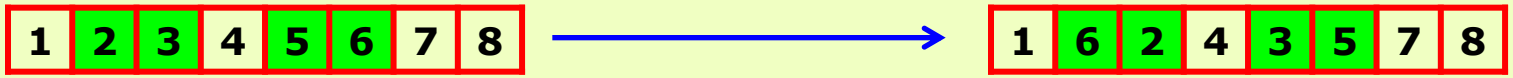
- Mutaţie prin inversare
  - Ideea de bază
    - Se aleg aleator 2 gene şi se inversează ordinea genelor situate între ele (substringul dintre gene)



Algoritmi evolutivi – algoritm
Proiectare – mutația (reprez. permutare)

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

- Mutaţie prin amestec (*scramble mutation*)
  - Ideea de bază
    - Se alege aleator un subşir (continuu sau discontinuu) de gene şi se rearanjează acele gene



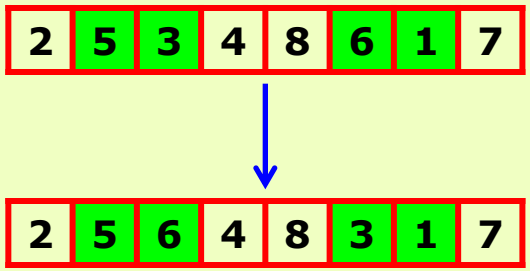
Algoritmi evolutivi – algoritm
Proiectare – mutația (reprez. permutare)

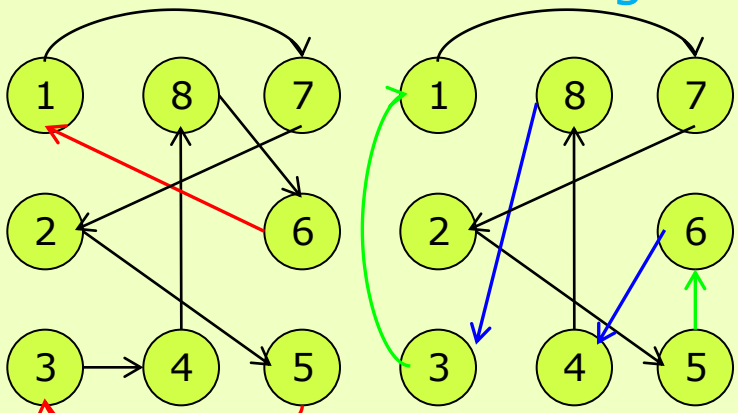
- Un cromozom $c=(g_1,g_2,...,g_L)$ cu $g_i \neq g_i$ pentru orice $i \neq i$ devine $c'=(g_1',g_2',...,g_L')$, unde $g_i, g_i' \in \{val_1, val_2,...,val_L\}$, pt. $i=1,2,...,L$ a.î. $g_i' \neq g_i'$ pentru orice $i \neq i$.

Mutaţie k-opt

- Ideea de bază
  - Se aleg 2 substringuri disjuncte şi de lungime *k*
  - Se interchimbă 2 elemente ale acestor substringuri de gene

k=2





Algoritmi evolutivi – algoritm
Proiectare – mutația (reprez. reală)

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

- Mutaţie 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)

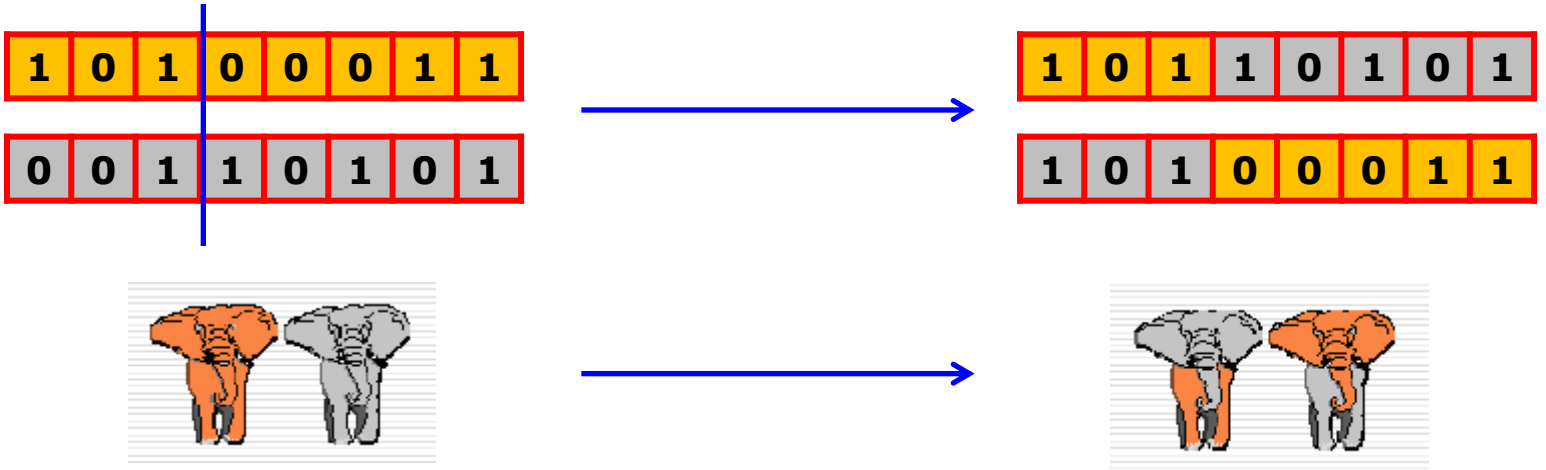
- Un cromozom $c=(g_1,g_2,...,g_L)$ devine $c'=(g_1',g_2',...,g_L')$, unde $g_i, g_i' \in [LI_i, LS_i]$, pt. $i=1,2,...,L$

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

Algoritmi evolutivi – algoritm
Proiectare - recombinarea

- Scop
  - Amestecarea informaţiilor preluate din părinţi

- Proprietăţi
  - Descendentul trebuie să moştenească ceva de la fiecare dintre părinţi
    - Alegerea informaţilor care se amestecă este aleatoare
  - Operator de exploatare probabilistică (p_c) a spaţiilor deja descoperite
  - Descendenţii pot să fie mai buni, la fel de buni sau mai slabi decât părinţii lor
  - Efectele sale se reduc pe măsură ce căutarea converge



Algoritmi evolutivi – algoritm Proiectare - recombinarea

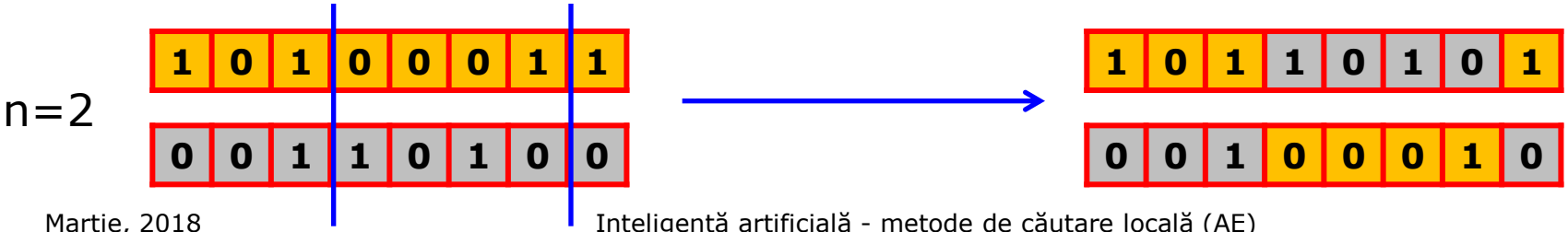
- Tipologie – în funcţie de reprezentarea indivizilor
  - Reprezentare binară şi întreagă
    - Cu puncte de tăietură
    - Uniformă
  - Reprezentare cu permutări
    - Încrucişare prin ordonare (versiunea 1 şi versiunea 2)
    - Încrucişare transformată parţial (Partially Mapped Crossover)
    - Încrucişare ciclică
    - Încrucişare bazată pe legături (muchii)
  - Reprezentare reală
    - Discretă
    - Intermediară (aritmetică)
      - Aritmetică singulară
      - Aritmetică simplă
      - Aritmetică completă
    - Geometrică
    - Încrucişare amestecată
    - Încrucişare binară simulată
  - Reprezentare cu arbori
    - Încrucişare de sub-arbori → într-un curs viitor

Algoritmi evolutivi – algoritm
Proiectare – recombinarea (reprez. binară şi întreagă)

- Din 2 cromozomi părinţi
  - $p_1=(g_1^1,g_2^1,\ldots,g_L^1)$ şi $p_2=(g_1^2,g_2^2,\ldots,g_L^2)$
- se obţin 2 descendenţi
  - $c_1=(g_1',g_2',\ldots,g_L')$ şi $c_2=(g_1'',g_2'',\ldots,g_L'')$,
  - unde $g_i^1,g_i^2, g_i', g_i'' \in \{0,1\} / \{val_1, val_2, \ldots, val_k\}$, pt. i=1,2,...,L

Încrucişare cu *n* puncte de tăietură

- Ideea de bază
  - Se aleg *n* puncte de tăietură ($n < L$)
  - Se taie cromozomii părinţi prin aceste puncte
  - Se lipesc părţile obţinute, alternând părinţii



Algoritmi evolutivi – algoritm
Proiectare – recombinarea (reprez. binară şi întreagă)

- Încrucişare cu *n* puncte de tăietură
  - Proprietăţi
    - Media valorilor codate de părinţi = media valorilor codate de descendenţi
      - Ex. Reprezentarea binară pe 4 biţi a numerelor întregi – XO cu n = 1 dupa 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) ➔ (val(p_1) + val(p_2))/2 = 23/2=11.5
        - val(d_1) = 9, val(d_2) = (14) ➔ (val(d_1) + val(d_2))/2 = 23/2=11.5
      - Ex. Reprezentare binară pe 4 biţi pentru problema rucsacului de capacitate K = 10 cu 4 obiecte de greutate şi 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 ➔ (val(p_1) + val(p_2))/2 = 26/2=13
        - val(d_1) = 10, val(d_2) = 16 ➔ (val(d_1) + val(d_2))/2 = 26/2=13
    - Probabilitatea apariţiei unui factor de răspândire $\beta \approx 1$ este mai mare decât probabilitatea oricărui alt factor

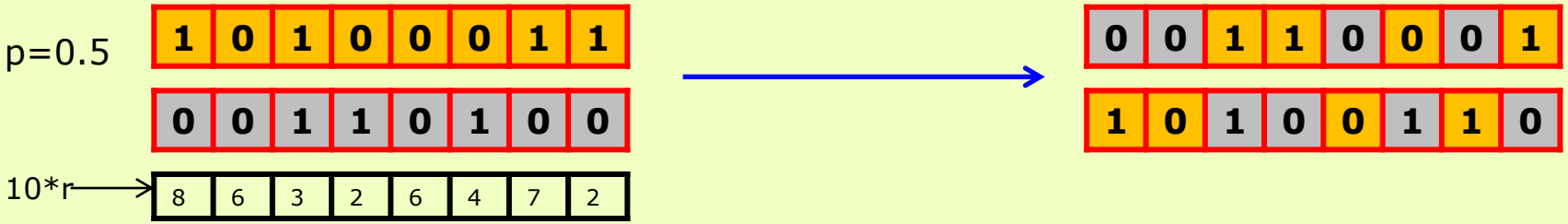
$$\beta = \left| \frac{val(d_1) - val(d_2)}{val(p_1) - val(p_2)} \right|$$

      - Încrucişare prin contracţie $\beta < 1$
        - Valorile descendenţilor se află între valorile părinţilor
      - Încrucişare prin extensie $\beta > 1$
        - Valorile părinţilor se află între valorile descendenţilor
      - Încrucişare staţionară $\beta = 1$
        - Valorile descendenţilor coincid cu valorile părinţilor

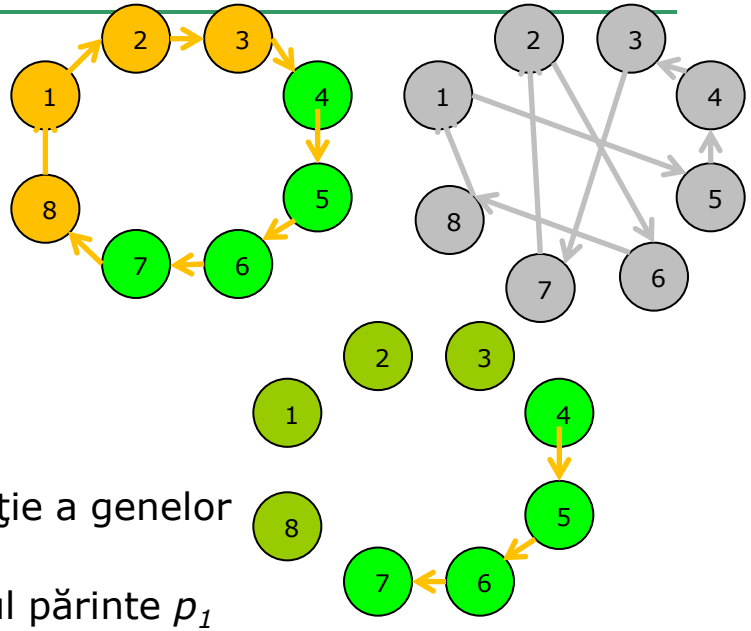
Algoritmi evolutivi – algoritm
Proiectare – recombinarea (reprez. binară şi întreagă)

- Din 2 cromozomi părinţi
  - $p_1=(g_1^1,g_2^1,\ldots,g_L^1)$ şi $p_2=(g_1^2,g_2^2,\ldots,g_L^2)$
- se obţin 2 descendenţi
  - $c_1=(g_1',g_2',\ldots,g_L')$ şi $c_2=(g_1'',g_2'',\ldots,g_L'')$,
  - unde $g_i^1,g_i^2, g_i', g_i'' \in \{0,1\} / \{val_1, val_2, \ldots, val_k\}$, pt. i=1,2,...,L

- Încrucişare uniformă
  - Ideea de bază
    - Fiecare genă a unui descendent provine dintr-un părinte ales aleator şi 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 şi c_2 va lua gena respectivă din p_2,
      - Altfel c_1 va lua gena respectivă din p_2 şi c_2 va lua gena respectivă din p_1

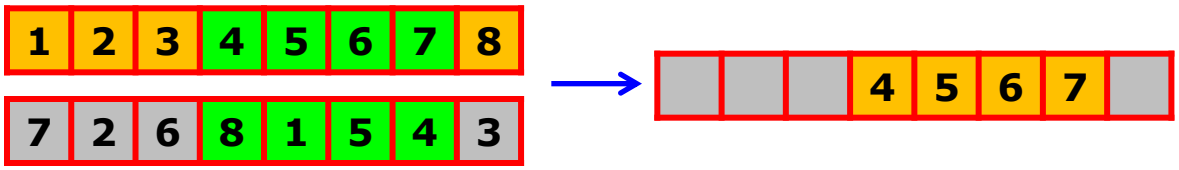


Algoritmi evolutivi – algoritm
Proiectare – recombinarea (reprez. permutare)



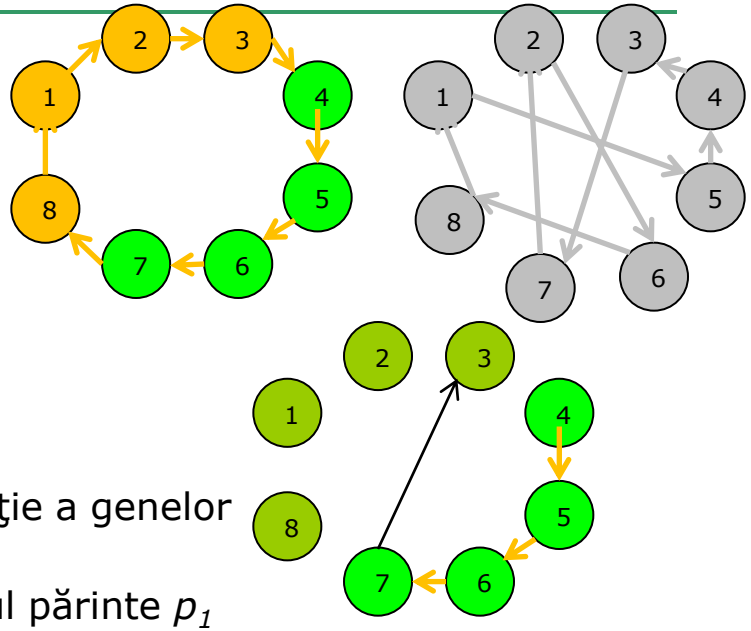
- Din 2 cromozomi părinţi
  - $p_1=(g_1^1,g_2^1,\ldots,g_L^1)$ şi $p_2=(g_1^2,g_2^2,\ldots,g_L^2)$
- se obţin 2 descendenţi
  - $c_1=(g_1',g_2',\ldots,g_L')$ şi $c_2=(g_1'',g_2'',\ldots,g_L'')$,
  - unde $g_i^1,g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, pt. i=1,2,…,L

- Încrucişare ordonată
  - Ideea de bază
    - Descendenţii păstrează ordinea de apariţie a genelor părinţilor
    - Se alege un substring de gene din primul părinte p_1
    - Se copiază substringul din p_1 în descendentul d_1 (pe poziţii corespondente)



Algoritmi evolutivi – algoritm
Proiectare – recombinarea (reprez. permutare)

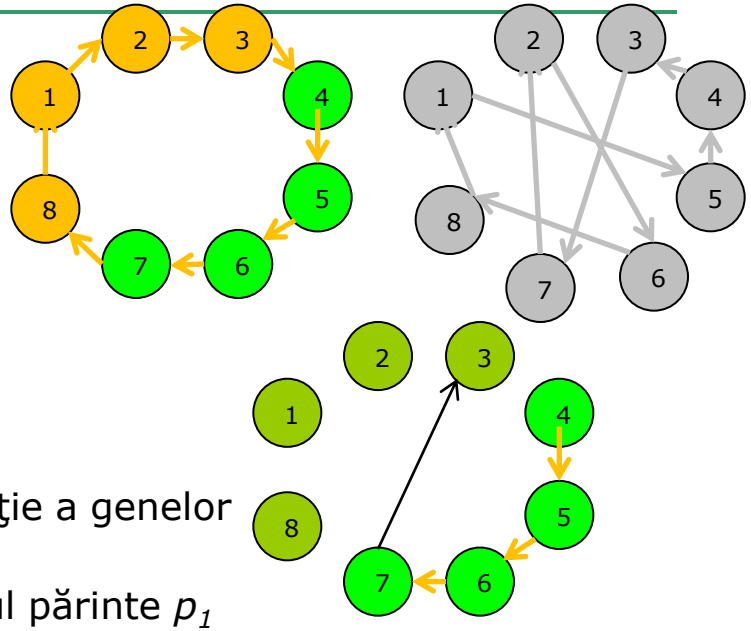
- Din 2 cromozomi părinţi
  - $p_1=(g_1^1,g_2^1,\ldots,g_L^1)$ şi $p_2=(g_1^2,g_2^2,\ldots,g_L^2)$
- se obţin 2 descendenţi
  - $c_1=(g_1',g_2',\ldots,g_L')$ şi $c_2=(g_1'',g_2'',\ldots,g_L'')$,
  - unde $g_i^1,g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, pt. i=1,2,…,L



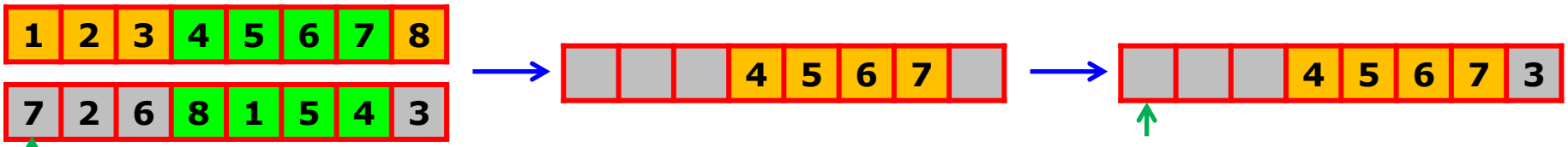
- Încrucişare ordonată
  - Ideea de bază
    - Descendenţii păstrează ordinea de apariţie a genelor părinţilor
    - Se alege un substring de gene din primul părinte p_1
    - Se copiază substringul din p_1 în descendentul d_1 (pe poziţii corespondente)
    - Se copiază genele din p_2 în descendentul d_1 astfel:
      - Începând cu prima poziţie de după terminarea substringului
      - Respectând ordinea genelor din p_2 şi
      - Re-luând genele de la prima poziţe (dacă s-a ajuns la sfârşit)

Algoritmi evolutivi – algoritm
Proiectare – recombinarea (reprez. permutare)

- Din 2 cromozomi părinţi
  - $p_1=(g_1^1,g_2^1,\ldots,g_L^1)$ şi $p_2=(g_1^2,g_2^2,\ldots,g_L^2)$
- se obţin 2 descendenţi
  - $c_1=(g_1',g_2',\ldots,g_L')$ şi $c_2=(g_1'',g_2'',\ldots,g_L'')$,
  - unde $g_i^1,g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, pt. i=1,2,…,L

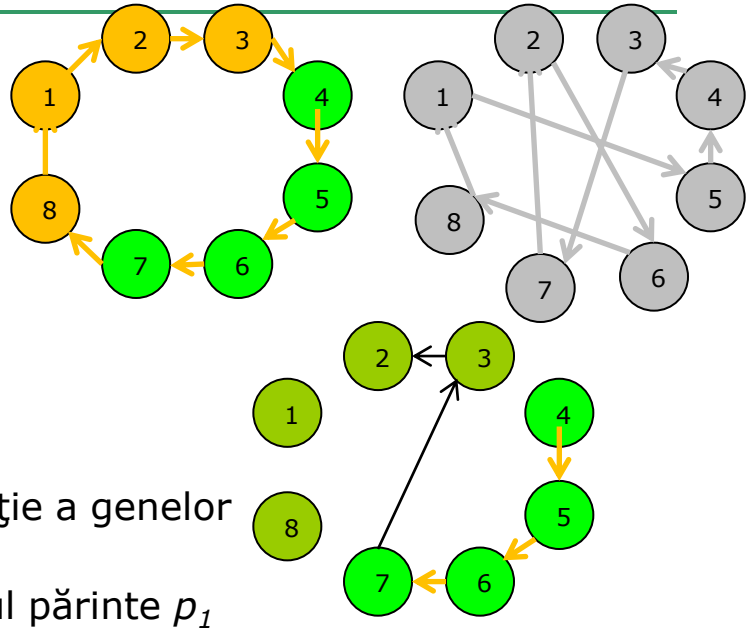


- Încrucişare ordonată
  - Ideea de bază
    - Descendenţii păstrează ordinea de apariţie a genelor părinţilor
    - Se alege un substring de gene din primul părinte p_1
    - Se copiază substringul din p_1 în descendentul d_1 (pe poziţii corespondente)
    - Se copiază genele din p_2 în descendentul d_1 astfel:
      - Începând cu prima poziţie de după terminarea substringului
      - Respectând ordinea genelor din p_2 şi
      - Re-luând genele de la prima poziţe (dacă s-a ajuns la sfârşit)

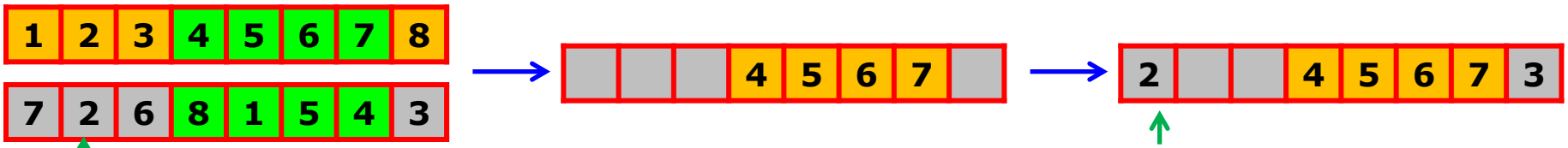


Algoritmi evolutivi – algoritm
Proiectare – recombinarea (reprez. permutare)

- Din 2 cromozomi părinţi
  - $p_1=(g_1^1,g_2^1,\ldots,g_L^1)$ şi $p_2=(g_1^2,g_2^2,\ldots,g_L^2)$
- se obţin 2 descendenţi
  - $c_1=(g_1',g_2',\ldots,g_L')$ şi $c_2=(g_1'',g_2'',\ldots,g_L'')$,
  - unde $g_i^1,g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, pt. i=1,2,…,L

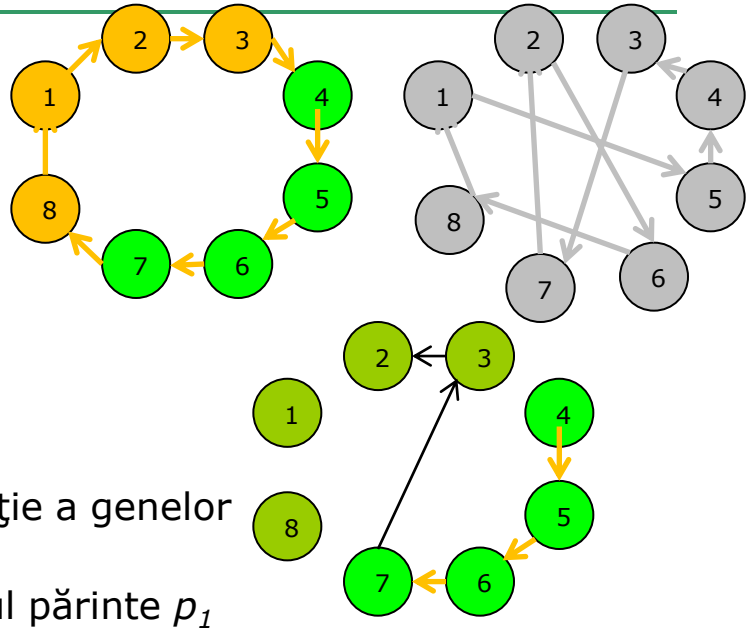


- Încrucişare ordonată
  - Ideea de bază
    - Descendenţii păstrează ordinea de apariţie a genelor părinţilor
    - Se alege un substring de gene din primul părinte p_1
    - Se copiază substringul din p_1 în descendentul d_1 (pe poziţii corespondente)
    - Se copiază genele din p_2 în descendentul d_1 astfel:
      - Începând cu prima poziţie de după terminarea substringului
      - Respectând ordinea genelor din p_2 şi
      - Re-luând genele de la prima poziţe (dacă s-a ajuns la sfârşit)



Algoritmi evolutivi – algoritm
Proiectare – recombinarea (reprez. permutare)

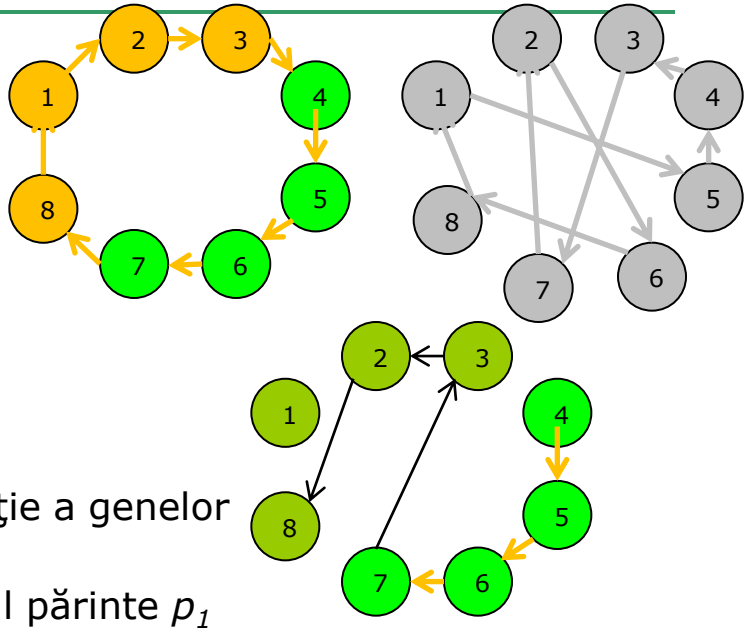
- Din 2 cromozomi părinţi
  - $p_1=(g_1^1,g_2^1,\ldots,g_L^1)$ şi $p_2=(g_1^2,g_2^2,\ldots,g_L^2)$
- se obţin 2 descendenţi
  - $c_1=(g_1',g_2',\ldots,g_L')$ şi $c_2=(g_1'',g_2'',\ldots,g_L'')$,
  - unde $g_i^1,g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, pt. i=1,2,...,L



- Încrucişare ordonată
  - Ideea de bază
    - Descendenţii păstrează ordinea de apariţie a genelor părinţilor
    - Se alege un substring de gene din primul părinte p_1
    - Se copiază substringul din p_1 în descendentul d_1 (pe poziţii corespondente)
    - Se copiază genele din p_2 în descendentul d_1 astfel:
      - Începând cu prima poziţie de după terminarea substringului
      - Respectând ordinea genelor din p_2 şi
      - Re-luând genele de la prima poziţe (dacă s-a ajuns la sfârşit)

Algoritmi evolutivi – algoritm
Proiectare – recombinarea (reprez. permutare)

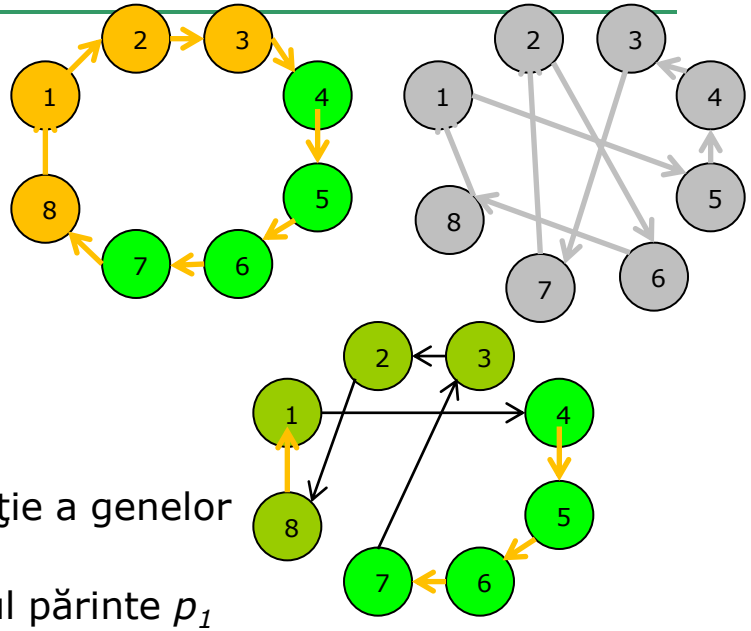
- Din 2 cromozomi părinţi
  - $p_1=(g_1^1,g_2^1,\ldots,g_L^1)$ şi $p_2=(g_1^2,g_2^2,\ldots,g_L^2)$
- se obţin 2 descendenţi
  - $c_1=(g_1',g_2',\ldots,g_L')$ şi $c_2=(g_1'',g_2'',\ldots,g_L'')$,
  - unde $g_i^1,g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, pt. i=1,2,...,L



- Încrucişare ordonată
  - Ideea de bază
    - Descendenţii păstrează ordinea de apariţie a genelor părinţilor
    - Se alege un substring de gene din primul părinte p_1
    - Se copiază substringul din p_1 în descendentul d_1 (pe poziţii corespondente)
    - Se copiază genele din p_2 în descendentul d_1 astfel:
      - Începând cu prima poziţie de după terminarea substringului
      - Respectând ordinea genelor din p_2 şi
      - Re-luând genele de la prima poziţe (dacă s-a ajuns la sfârşit)

Algoritmi evolutivi – algoritm
Proiectare – recombinarea (reprez. permutare)

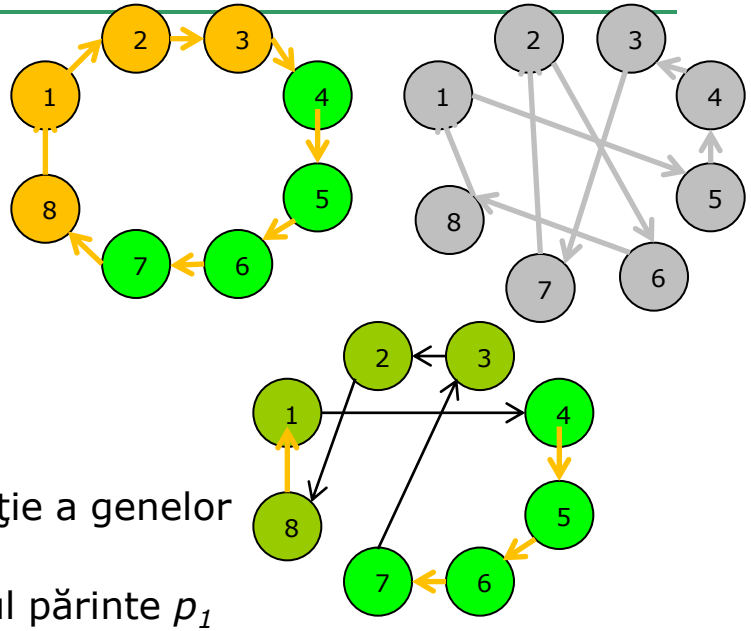
- Din 2 cromozomi părinţi
  - $p_1=(g_1^1,g_2^1,\ldots,g_L^1)$ şi $p_2=(g_1^2,g_2^2,\ldots,g_L^2)$
- se obţin 2 descendenţi
  - $c_1=(g_1',g_2',\ldots,g_L')$ şi $c_2=(g_1'',g_2'',\ldots,g_L'')$,
  - unde $g_i^1,g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, pt. i=1,2,...,L



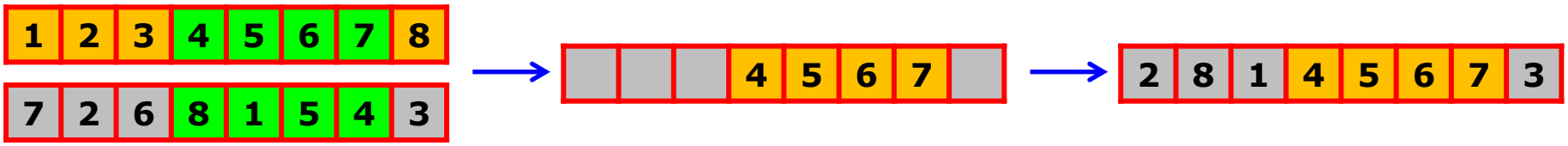
- Încrucişare ordonată
  - Ideea de bază
    - Descendenţii păstrează ordinea de apariţie a genelor părinţilor
    - Se alege un substring de gene din primul părinte p_1
    - Se copiază substringul din p_1 în descendentul d_1 (pe poziţii corespondente)
    - Se copiază genele din p_2 în descendentul d_1 astfel:
      - Începând cu prima poziţie de după terminarea substringului
      - Respectând ordinea genelor din p_2 şi
      - Re-luând genele de la prima poziţe (dacă s-a ajuns la sfârşit)

Algoritmi evolutivi – algoritm
Proiectare – recombinarea (reprez. permutare)

- Din 2 cromozomi părinţi
  - $p_1=(g_1^1,g_2^1,\ldots,g_L^1)$ şi $p_2=(g_1^2,g_2^2,\ldots,g_L^2)$
- se obţin 2 descendenţi
  - $c_1=(g_1',g_2',\ldots,g_L')$ şi $c_2=(g_1'',g_2'',\ldots,g_L'')$,
  - unde $g_i^1,g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, pt. i=1,2,...,L



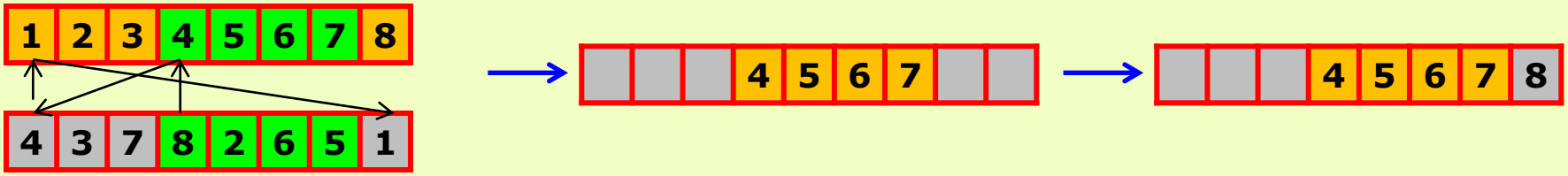
- Încrucişare ordonată
  - Ideea de bază
    - Descendenţii păstrează ordinea de apariţie a genelor părinţilor
    - Se alege un substring de gene din primul părinte p_1
    - Se copiază substringul din p_1 în descendentul d_1 (pe poziţii corespondente)
    - Se copiază genele din p_2 în descendentul d_1 astfel:
      - Începând cu prima poziţie de după terminarea substringului
      - Respectând ordinea genelor din p_2 şi
      - Re-luând genele de la prima poziţe (dacă s-a ajuns la sfârşit)
    - Se reia procedeul pentru al doilea descendent d_2.



Algoritmi evolutivi – algoritm
Proiectare – recombinarea (reprez. permutare)

- Din 2 cromozomi părinţi
  - $p_1=(g_1^1,g_2^1,\ldots,g_L^1)$ şi $p_2=(g_1^2,g_2^2,\ldots,g_L^2)$
- se obţin 2 descendenţi
  - $c_1=(g_1',g_2',\ldots,g_L')$ şi $c_2=(g_1'',g_2'',\ldots,g_L'')$,
  - unde $g_i^1,g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, pt. $i=1,2,\ldots,L$

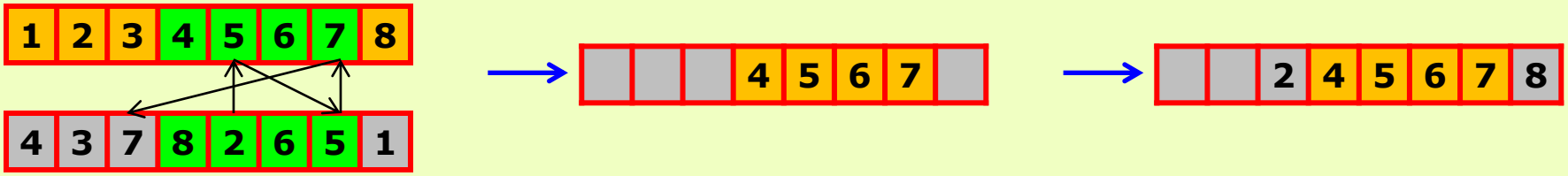
- Încrucişare parţial transformată (*partially mapped XO*)
  - Ideea de bază
    - Se alege un substring de gene din primul părinte p_1
    - Se copiază substringul din p_1 în descendentul d_1 (pe poziţii corespondente)
    - Se iau pe rând elementele i din substringul din p_2 care nu apar în substringul din p_1 şi se determină care element j a fost copiat în locul lui din p_1
    - Se plasează i în d_1 în poziţia 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 părinţii



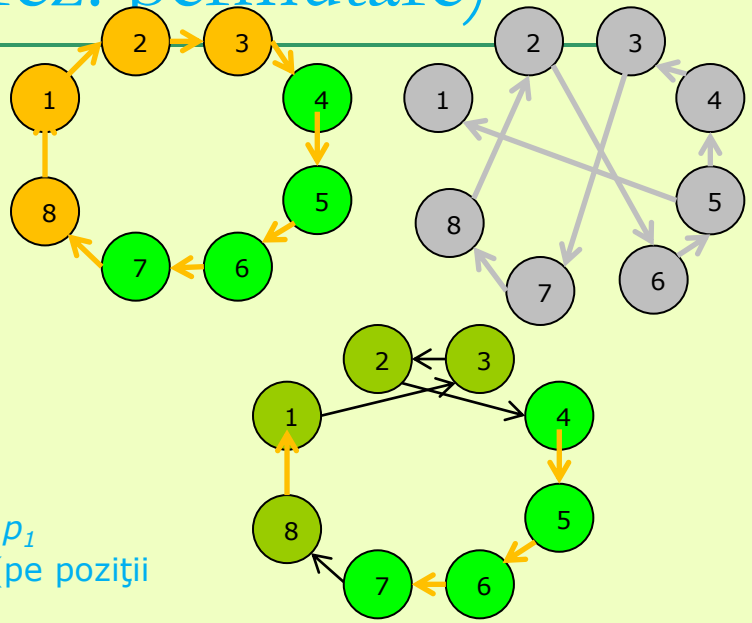
Algoritmi evolutivi – algoritm
Proiectare – recombinarea (reprez. permutare)

- Din 2 cromozomi părinţi
  - $p_1=(g_1^1,g_2^1,\ldots,g_L^1)$ şi $p_2=(g_1^2,g_2^2,\ldots,g_L^2)$
- se obţin 2 descendenţi
  - $c_1=(g_1',g_2',\ldots,g_L')$ şi $c_2=(g_1'',g_2'',\ldots,g_L'')$,
  - unde $g_i^1,g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, pt. $i=1,2,\ldots,L$

- Încrucişare parţial 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 poziţii corespondente)
    - Se iau pe rând elementele i din substringul din p_2 care nu apar în substringul din p_1 şi se determină care element j a fost copiat în locul lui din p_1
    - Se plasează i în d_1 în poziţia 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 părinţii

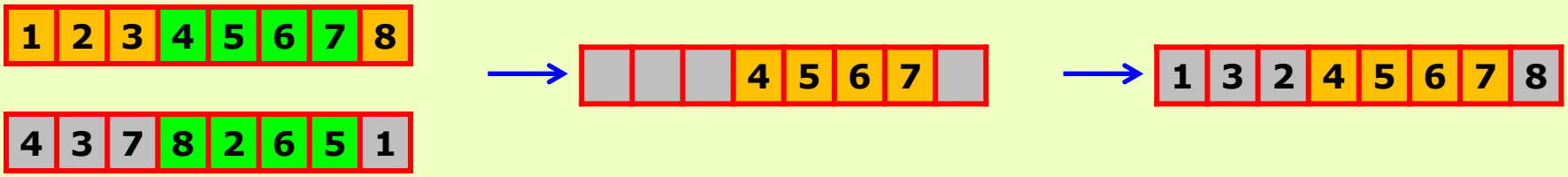


Algoritmi evolutivi – algoritm
Proiectare – recombinarea (reprez. permutare)



- Din 2 cromozomi părinţi
  - $p_1=(g_1^1,g_2^1,\ldots,g_L^1)$ şi $p_2=(g_1^2,g_2^2,\ldots,g_L^2)$
- se obţin 2 descendenţi
  - $c_1=(g_1',g_2',\ldots,g_L')$ şi $c_2=(g_1'',g_2'',\ldots,g_L'')$,
  - unde $g_i^1,g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, pt. i=1,2,...,L

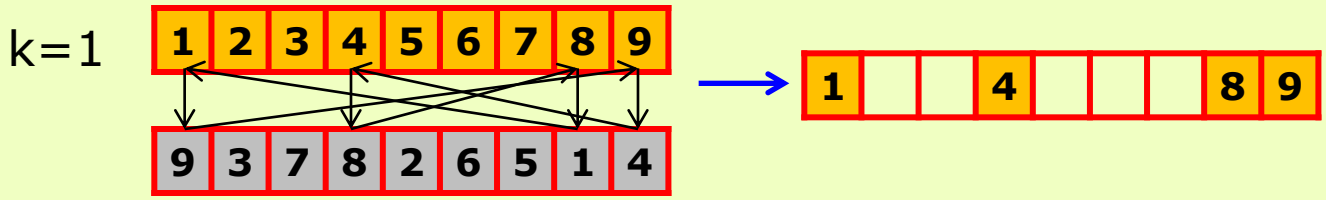
- Încrucişare parţial transformată
  - Ideea de bază
    - Se alege un substring de gene din primul părinte p_1
    - Se copiază substringul din p_1 în descendentul d_1 (pe poziţii corespondente)
    - Se iau pe rând elementele i din substringul din p_2 care nu apar în substringul din p_1 şi se determină care element j a fost copiat în locul lui din p_1
    - Se plasează i în d_1 în poziţia 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 părinţii



Algoritmi evolutivi – algoritm
Proiectare – recombinarea (reprez. permutare)

- Din 2 cromozomi părinţi
  - $p_1=(g_1^1,g_2^1,\ldots,g_L^1)$ şi $p_2=(g_1^2,g_2^2,\ldots,g_L^2)$
- se obţin 2 descendenţi
  - $c_1=(g_1',g_2',\ldots,g_L')$ şi $c_2=(g_1'',g_2'',\ldots,g_L'')$,
  - unde $g_i^1,g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, pt. i=1,2,...,L

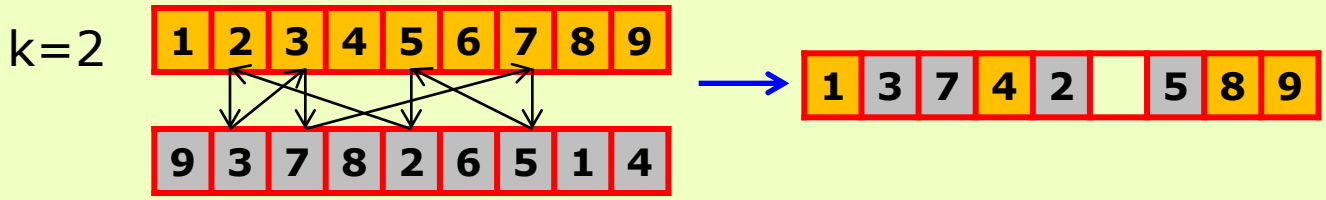
- Încrucişare ciclică
  - Ideea de bază
    1. iniţial k = 1
    2. Se formează un ciclu
       - Se adaugă în ciclu gena de pe poziţia k din p1 (g_k^1)
       - Se consideră gena de pe poziţia k din p2 (g_k^2)
       - Se alege gena din p1 cu valoarea egală cu g_k^2 (g_r^1) şi se include în ciclu
       - Se consideră gena de pe poziţia r din p2 (g_r^2)
       - Se repetă paşii anteriori până când se ajunge la gena de pe poziţia k din p1
    3. Se copiază genele din ciclu în d1 (respectând poziiţiile pe care apar în p1)
    4. Se incrementează k şi se formează un nou ciclu dar cu genele din p2
    5. Se copiază genele din ciclu în d1 (respectând poziiţiile pe care apar în p2)
    6. Se repetă paşii 2-5 până când k = L



Algoritmi evolutivi – algoritm
Proiectare – recombinarea (reprez. permutare)

- Din 2 cromozomi părinţi
  - $p_1=(g_1^1,g_2^1,\ldots,g_L^1)$ şi $p_2=(g_1^2,g_2^2,\ldots,g_L^2)$
- se obţin 2 descendenţi
  - $c_1=(g_1',g_2',\ldots,g_L')$ şi $c_2=(g_1'',g_2'',\ldots,g_L'')$,
  - unde $g_i^1,g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, pt. $i=1,2,\ldots,L$

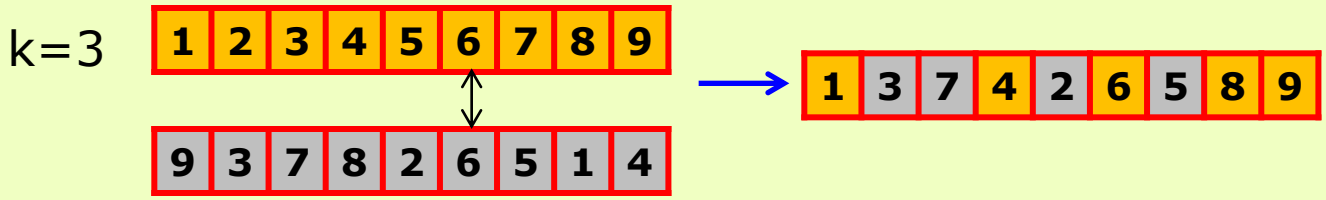
- Încrucişare ciclică
  - Ideea de bază
    1. iniţial k = 1
    2. Se formează un ciclu
       - Se adaugă în ciclu gena de pe poziţia k din p1 (g_k^1)
       - Se consideră gena de pe poziţia k din p2 (g_k^2)
       - Se alege gena din p1 cu valoarea egală cu g_k^2 (g_r^1) şi se include în ciclu
       - Se consideră gena de pe poziţia r din p2 (g_r^2)
       - Se repetă paşii anteriori până când se ajunge la gena de pe poziţia k din p1
    3. Se copiază genele din ciclu în d1 (respectând poziiţiile pe care apar în p1)
    4. Se incrementează k şi se formează un nou ciclu dar cu genele din p2
    5. Se copiază genele din ciclu în d1 (respectând poziiţiile pe care apar în p2)
    6. Se repetă paşii 2-5 până când k = L



Algoritmi evolutivi – algoritm
Proiectare – recombinarea (reprez. permutare)

- Din 2 cromozomi părinţi
  - $p_1=(g_1^1,g_2^1,\ldots,g_L^1)$ şi $p_2=(g_1^2,g_2^2,\ldots,g_L^2)$
- se obţin 2 descendenţi
  - $c_1=(g_1',g_2',\ldots,g_L')$ şi $c_2=(g_1'',g_2'',\ldots,g_L'')$,
  - unde $g_i^1,g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, pt. $i=1,2,\ldots,L$

- Încrucişare ciclică
  - Ideea de bază
    1. iniţial k = 1
    2. Se formează un ciclu
       - Se adaugă în ciclu gena de pe poziţia k din p1 (g_k^1)
       - Se consideră gena de pe poziţia k din p2 (g_k^2)
       - Se alege gena din p1 cu valoarea egală cu g_k^2 (g_r^1) şi se include în ciclu
       - Se consideră gena de pe poziţia r din p2 (g_r^2)
       - Se repetă paşii anteriori până când se ajunge la gena de pe poziţia k din p1
    3. Se copiază genele din ciclu în d1 (respectând poziiţiile pe care apar în p1)
    4. Se incrementează k şi se formează un nou ciclu dar cu genele din p2
    5. Se copiază genele din ciclu în d1 (respectând poziiţiile pe care apar în p2)
    6. Se repetă paşii 2-5 până când k = L



Algoritmi evolutivi – algoritm
Proiectare – recombinarea (reprez. permutare)

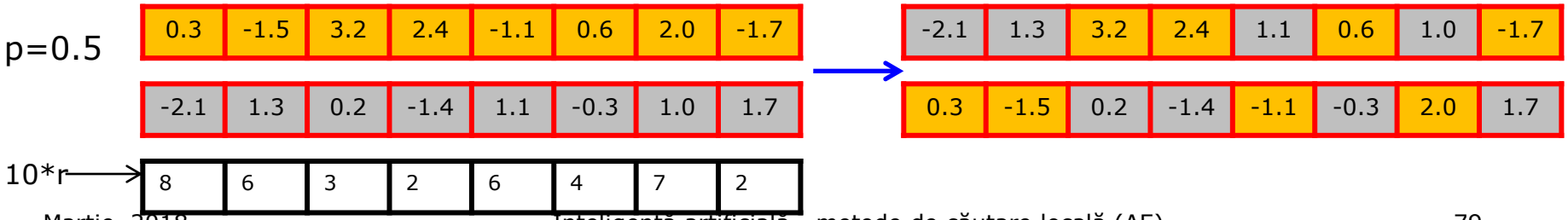
- Din 2 cromozomi părinţi
  - $p_1=(g_1^1,g_2^1,\ldots,g_L^1)$ şi $p_2=(g_1^2,g_2^2,\ldots,g_L^2)$
- se obţin 2 descendenţi
  - $c_1=(g_1',g_2',\ldots,g_L')$ şi $c_2=(g_1'',g_2'',\ldots,g_L'')$,
  - unde g_i^1,g_i^2, g_i', $g_i'' \in [LI_i, LS_i]$, pt. i=1,2,...,L

- Încrucişare 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 (reprez. reală)

- Din 2 cromozomi părinţi
  - $p_1=(g_1^1,g_2^1,\ldots,g_L^1)$ şi $p_2=(g_1^2,g_2^2,\ldots,g_L^2)$
- se obţin 2 descendenţi
  - $c_1=(g_1',g_2',\ldots,g_L')$ şi $c_2=(g_1'',g_2'',\ldots,g_L'')$,
  - unde $g_i^1,g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, pt. i=1,2,...,L

- Încrucişare discretă
  - Ideea de bază
    - Fiecare genă a unui descendent este luată (cu aceeaşi probabilitate, *p = 0.5*) dintr-unul din părinţi
    - Similar încrucişării uniforme de la reprezentarea binară/întreagă
    - Nu se modifică valorile efective ale genelor (nu se creează informaţie nouă)



Algoritmi evolutivi – algoritm
Proiectare – recombinarea (reprez. reală)

- Din 2 cromozomi părinţi
  - $p_1=(g_1^1,g_2^1,\ldots,g_L^1)$ şi $p_2=(g_1^2,g_2^2,\ldots,g_L^2)$
- se obţin 2 descendenţi
  - $c_1=(g_1',g_2',\ldots,g_L')$ şi $c_2=(g_1'',g_2'',\ldots,g_L'')$,
  - unde $g_i^1,g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, pt. i=1,2,...,L

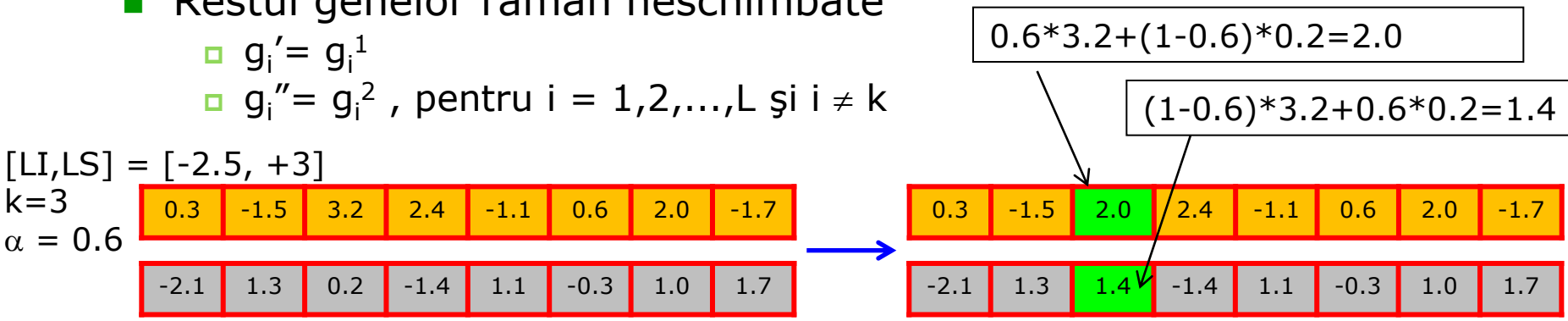
- **Încrucişare intermediară (aritmetică)**
  - Ideea de bază
    - Se creează copii aflaţi (ca valoare) între părinţi ➔ încrucişare aritmetică
      - $z_i = \alpha x_i + (1 - \alpha) y_i$ unde α : $0 \leq \alpha \leq 1$.
    - Parametrul α poate fi:
      - Constant → încrucişare aritmetică uniformă
      - Variabil → ex. dependent de vârsta populaţiei
      - Aleator pt fiecare încrucişare produsă
    - Apar noi valori ale genelor

- Tipologie
  - Încrucişare aritmetică singulară
  - Încrucişare aritmetică simplă
  - Încrucişare aritmetică completă

Algoritmi evolutivi – algoritm
Proiectare – recombinarea (reprez. reală)

- Din 2 cromozomi părinţi
  - $p_1=(g_1^1,g_2^1,\ldots,g_L^1)$ şi $p_2=(g_1^2,g_2^2,\ldots,g_L^2)$
- se obţin 2 descendenţi
  - $c_1=(g_1',g_2',\ldots,g_L')$ şi $c_2=(g_1'',g_2'',\ldots,g_L'')$,
  - unde $g_i^1,g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, pt. $i=1,2,\ldots,L$

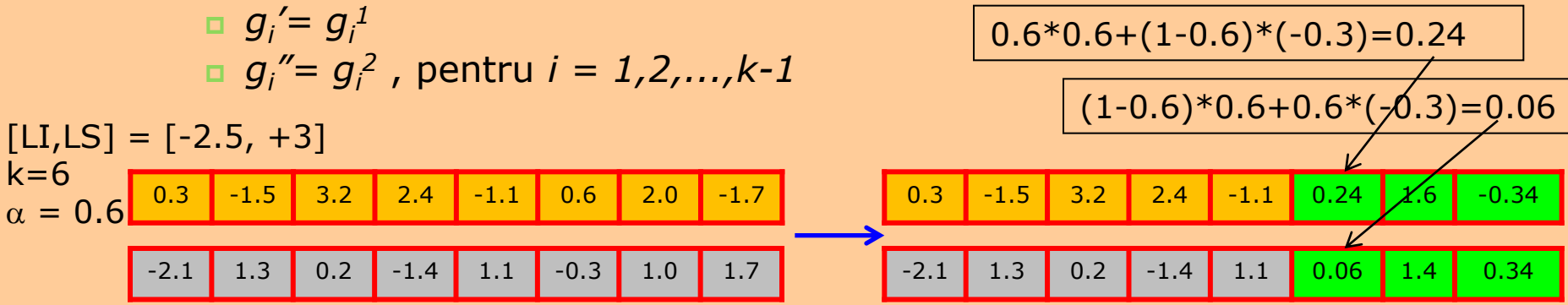
- Încrucişare intermediară (aritmetică) singulară
  - Se alege câte o genă (de acelaşi index *k*) din cei doi părinţi şi se combină
    - $g_k'= \alpha g_k^1 + (1-\alpha)g_k^2$
    - $g_k''= (1-\alpha)g_k^1 + \alpha g_k^2$
  - Restul genelor rămân neschimbate
    - $g_i'= g_i^1$
    - $g_i''= g_i^2$, pentru $i = 1,2,\ldots,L$ şi $i \neq k$



Algoritmi evolutivi – algoritm
Proiectare – recombinarea (reprez. reală)

- Din 2 cromozomi părinţi
  - $p_1=(g_1^1,g_2^1,\ldots,g_L^1)$ şi $p_2=(g_1^2,g_2^2,\ldots,g_L^2)$
- se obţin 2 descendenţi
  - $c_1=(g_1',g_2',\ldots,g_L')$ şi $c_2=(g_1'',g_2'',\ldots,g_L'')$,
  - unde $g_i^1,g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, pt. $i=1,2,\ldots,L$

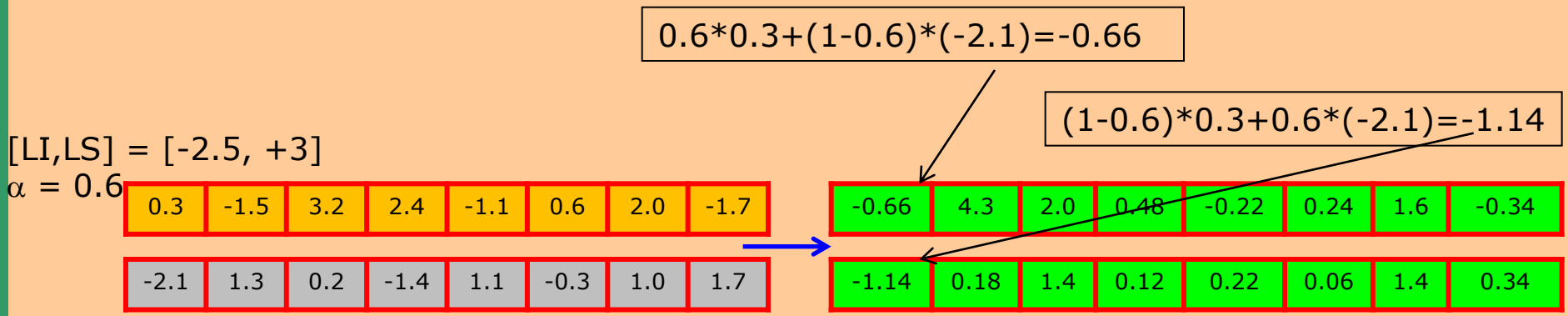
- Încrucişare intermediară (aritmetică) simplă
  - Se alege o poziţie *k* şi se combină toate genele de după acea poziţie
    - $g_i' = \alpha g_i^1 + (1-\alpha) g_i^2$
    - $g_i'' = (1-\alpha) g_i^1 + \alpha g_i^2$, pentru $i=k, k+1, \ldots, L$
  - Genele de pe poziţii < *k* rămân neschimbate
    - $g_i' = g_i^1$
    - $g_i'' = g_i^2$, pentru $i = 1,2,\ldots,k-1$



Algoritmi evolutivi – algoritm
Proiectare – recombinarea (reprez. reală)

- Din 2 cromozomi părinţi
  - $p_1=(g_1^1,g_2^1,\ldots,g_L^1)$ şi $p_2=(g_1^2,g_2^2,\ldots,g_L^2)$
- se obţin 2 descendenţi
  - $c_1=(g_1',g_2',\ldots,g_L')$ şi $c_2=(g_1'',g_2'',\ldots,g_L'')$,
  - unde $g_i^1,g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, pt. i=1,2,...,L

- **Încrucişare intermediară (aritmetică) completă**
  - Toate genele (de pe poziţii 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,...,L



Algoritmi evolutivi – algoritm
Proiectare – recombinarea (reprez. reală)

- Din 2 cromozomi părinţi
  - $p_1=(g_1^1,g_2^1,\ldots,g_L^1)$ şi $p_2=(g_1^2,g_2^2,\ldots,g_L^2)$
- se obţin 2 descendenţi
  - $c_1=(g_1',g_2',\ldots,g_L')$ şi $c_2=(g_1'',g_2'',\ldots,g_L'')$,
  - unde $g_i^1,g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, pt. i=1,2,...,L

- Încrucişare geometrică
  - Ideea de bază
    - Fiecare genă a unui descendent reprezintă produsul genelor părinţilor, fiecare cu un anumit exponent ω, respectiv 1-ω (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]

ω = 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 (reprez. reală)

- Din 2 cromozomi părinţi
  - $p_1=(g_1^1,g_2^1,\ldots,g_L^1)$ şi $p_2=(g_1^2,g_2^2,\ldots,g_L^2)$
- se obţine 1 descendent
  - $c_1=(g_1',g_2',\ldots,g_L')$,
  - unde $g_i^1, g_i^2, g_i' \in [LI_i, LS_i]$, pt. $i=1,2,\ldots,L$
- Încrucişare 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 – recombinarea (reprez. reală)

- Din 2 cromozomi părinţi
  - $p_1=(g_1^1,g_2^1,\ldots,g_L^1)$ şi $p_2=(g_1^2,g_2^2,\ldots,g_L^2)$
- se obţin 2 descendenţi
  - $c_1=(g_1',g_2',\ldots,g_L')$ şi $c_2=(g_1'',g_2'',\ldots,g_L'')$,
  - unde $g_i^1,g_i^2, g_i', g_i'' \in [LI_i, LS_i]$, pt. $i=1,2,\ldots,L$

- Încrucişare binară simulată
  - Ideea de bază
    - Fiecare genă a unui descendent reprezintă o combinaţie a genelor părinţilor

$$d_1=\frac{p_1+p_2}{2}-\beta\frac{p_2-p_1}{2},\quad d_2=\frac{p_1+p_2}{2}+\beta\frac{p_2-p_1}{2}$$

    - a.î. să se respecte cele 2 proprietăţi de la încrucişarea cu n puncte de tăietură (pt. reprezentarea binară)
      - media valorilor codate în părinţi = media valorilor codate în descendenţi
      - probabilitatea apariţiei unui factor de răspândire $\beta \approx 1$ este mai mare decât a oricărui alt factor

Algoritmi evolutivi – algoritm
Proiectare – recombinarea

- Recombinarea multiplă
  - Bazată pe frecvenţa valorilor din părinţi (încrucişare uniformă generală)
  - Bazată pe segmentare şi recombinare (încrucişare generală cu puncte de tăietură diagonală)
  - Bazată pe operaţii numerice specifice valorilor reale (încrucişare bazată pe centrul de masă, încrucişare 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 folosiţi ambii operatori
    - Fiecare având alt rol
    - Sunt posibili AE doar cu mutaţie, dar nu sunt posibili AE doar cu încrucişare
- Aspecte ale căutării:
  - Explorare → descoperirea regiunilor promiţătoare ale spaţiului de căutare (acumulând informaţie utilă despre problemă)
  - Exploatare → optimizarea într-o regiune promiţătoare (folosind informaţia existentă)
  - Trebuie să existe cooperare şi competiţie între aceste 2 aspecte
- Încrucişarea
  - Operator exploatativ, realizând un mare salt într-o regiune undeva între regiunile asociate părinţilor
    - Efectele exploatative se reduc pe măsură ce AE converge
  - Operator binar (n-ar) care poate combina informaţia din 2 (sau mai mulţi) părinţi
  - Operator care nu schimbă frecvenţa valorilor din cromozomi la nivelul întregii populaţii
- Mutaţia
  - Operator explorativ, realizând mici diversiuni aleatoare, rămânând în regiunea apropiată părintelui
    - Evadarea din optimele locale
  - Operator care poate introduce informaţie genetică nouă
  - Operator care schimbă frecvenţa valorilor din cromozomi la nivelul întregii populaţii

Algoritmi evolutivi – algoritm
Proiectare – criteriu de oprire

- Stabilirea unui criteriu de stop
  - S-a identificat soluţia optimă
  - S-au epuizat resursele fizice
    - S-a efectuat un anumit număr de evaluaări ale funcţiei de fitness
  - S-au epuizat resursele utilizatorului (timp, răbdare)
    - S-au "născut" câteva generaţii fără îmbunătăţiri

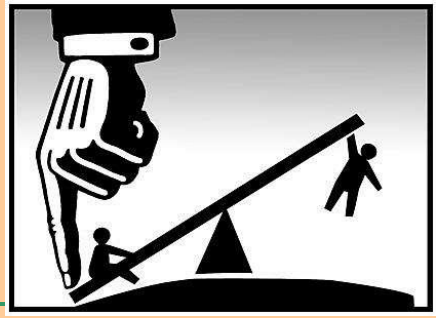
Algoritmi evolutivi - algoritm

- Evaluarea performanţelor unui AE
  - După mai multe rulări se calculează:
    - Măsuri statistice
      - media soluţiilor,
      - mediana soluţiilor,
      - cea mai bună soluţie,
      - cea mai slabă soluţie,
      - deviaţia standard – pentru comparabilitate
    - Calculate pentru un număr suficient de mare de rulări independente

Algoritmi evolutivi

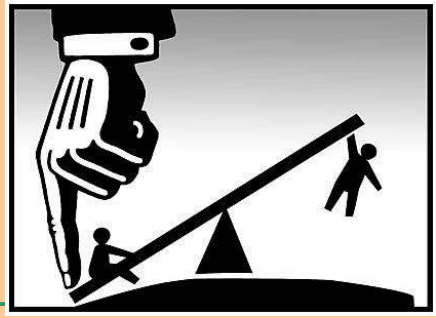
- Analiza complexităţii
  - Partea cea mai costisitoare → calculul fitnessului

Algoritmi evolutivi



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

Algoritmi evolutivi



- Dezavantaje
  - Timp de rulare îndelungat

Algoritmi evolutivi

- Aplicaţii
  - Proiectări vehicule
    - Componenţa materialelor
    - Forma vehiculelor
  - Proiectări inginereşti
    - Optimizarea structurală şi organizatorică a construcţiilor (clădiri, roboţi, sateliţi, turbine)
  - Robotică
    - Optimizarea proiectării, funcţionării componentelor
  - Evoluare de hardware
    - Optimizarea de circuite digitale
  - Optimizarea telecomunicaţiilor
  - Generarea de glume şi jocuri de cuvinte
  - Invenţii biomimetice (inspirate de arhitecturi naturale)
  - Rutări pentru trafic şi transporturi
  - Jocuri de calculator
  - Criptări
  - Profilul expresiv al genelor
  - Analiza chimcă a cinecticii
  - Strategii financiare şi marketing

Algoritmi evolutivi

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

Cursul următor

A. Scurtă introducere în Inteligenţa 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 învaţă singure
  - Arbori de decizie
  - Reţele neuronale artificiale
  - Maşini cu suport vectorial
  - Algoritmi evolutivi
- Sisteme bazate pe reguli
- Sisteme hibride

Cursul următor – Materiale de citit şi legături utile

- capitolul 16 din *C. Groşan, 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)

- Informaţiile prezentate au fost colectate din diferite surse de pe internet, precum şi din cursurile de inteligenţă artificială ţinute în anii anteriori de către:
  - Conf. Dr. Mihai Oltean – www.cs.ubbcluj.ro/~moltean
  - Lect. Dr. Crina Groşan - www.cs.ubbcluj.ro/~cgrosan
  - Prof. Dr. Horia F. Pop - www.cs.ubbcluj.ro/~hfpop