



**BABEŞ-BOLYAI UNIVERSITY**  
Faculty of Mathematics and Computer Science



# Inteligentă Artificială

**Camelia Chira**

[camelia.chira@ubbcluj.ro](mailto:camelia.chira@ubbcluj.ro)

# Obiective

- Rezolvarea problemelor reale cu metode inteligente
  - Intelegerea si modelarea problemei
  - Utilizarea / adaptarea unei metode inteligente pentru rezolvarea problemei
- Dezvoltarea de algoritmi si aplicatii pentru rezolvarea unor probleme complexe
  - Orice limbaj de programare
- **Modele din sfera IA**
  - Cautare locala
  - Simulated Annealing, Tabu search
  - Algoritmi Evolutivi
  - Algoritmi de tip Swarm Intelligence

# Curs Inteligenta Artificiala (IA): Scop

- A intlege in ce consta IA
  - Scop
  - Abilitati
  - Metodologie
  - Algoritmi
  - Aplicativitate
- A acumula informatii despre metode noi de rezolvare a problemelor prin:
  - Dezvoltarea de aplicatii inteligente
  - Introducerea conceptelor si tehnicilor de baza din IA
  - Intelegerea problemelor si dificultatilor intalnite in rezolvarea lor
  - Cunoasterea avantajelor si dezavantajelor unei anumite tehnici de rezolvare a problemelor
  - Exprimarea unei opinii critice asupra a ceea ce poate IA sa faca

# Bibliografie

- Z. Michalewicz, D. B. Fogel, How to solve it: Modern Heuristics, 2nd edition, Springer, 2004.
- S. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, Prentice Hall, 1995.
- C. Grosan, A. Abraham, Intelligent Systems: A Modern Approach, Springer, 2011.
- El-Ghazali Talbi, Metaheuristics - From Design to Implementation, Wiley 2009.
- M. Mitchell, An introduction to genetic algorithms, MIT Press, 1996.
- Back T, Fogel D.B, Michalewicz Z.; *Evolutionary Computation I. Basic Algorithms and Operators*, IOP Publ., 2000
- Back T, Fogel D.B, Michalewicz Z.; *Evolutionary Computation II. Advanced Algorithms and Operators*, IOP Publ., 2000
- A. E. Eiben, J.E. Smith, Introduction to Evolutionary Computing. Springer, 2003.
- D. E. Goldberg, Genetic Algorithms in Search, Optimization and Machine Learning, Addison-Wesley, 1989.
- K. A. De Jong, Evolutionary Computation: A Unified Approach. MIT Press, Cambridge, MA, 2006.

# Organizare

- **Structura**

- 12 cursuri
- 6 laboratoare
- 6 seminarii

- **Evaluare**

- Teme laborator
- Examen



<http://www.cs.ubbcluj.ro/~cchira>

MS Teams

# Evaluare

- **Punctaj total posibil 1100 puncte**

- **Teme Laborator - 50%**

P1 = max. 550 puncte

*4 teme laborator*

- **Examen - 50%**

P2 = max. 550 puncte

*Examen scris: intrebari grila si deschise  
sau*

*Proiect*

- **Nota finala se calculeaza in functie de punctajul total P = P1+P2**



Punctaj P	Nota
< 500	Nepromovat
[500, 599]	5
[600, 699]	6
[700, 799]	7
[800, 899]	8
[900, 999]	9
>= 1000	10

# Continut curs

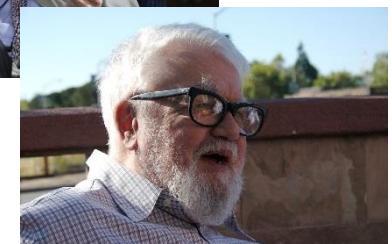
- Scurta introducere in IA
- Modelarea si rezolvarea problemelor, optimizare
- Algoritmi de cautare locala
  - Hill-climbing
- Algoritmi de tip single-point
  - Simulated Annealing
  - Tabu Search
- Algoritmi bazati pe populatii
  - Algoritmi Evolutivi
  - Algoritmi bazati pe inteligenta de grup (Swarm Intelligence)
- Modele hibride

# Ce este IA?

- Intrebare dificila
- AI is a branch of science which deals with helping machines find solutions to complex problems in a more human-like fashion.
- Pe scurt: determinarea mașinilor de a efectua lucruri inteligente

- **Minsky si McCarthy, 1950s:**

*"any task performed by a program or a machine that, if a human carried out the same activity, we would say the human had to apply intelligence to accomplish the task"*



- **Strong AI**

- Calculatoarele pot fi programate să gândească la un nivel cel puțin egal cu cel uman și chiar să fie conștiente de acțiunile lor

- **Weak AI**

- Calculatoarele pot efectua anumite sarcini de gândire – ceea ce deja se întâmplă

# De ce avem nevoie de IA?

- Optimizare si planificare
- Sisteme de recomandare
- Ordonarea paginilor web
- Recunoașterea/analiza vocii
- Recunoașterea imaginilor, scrisului de mână
- Traducerea automată
- Diagnosticare medicală
- Planificarea sarcinilor
- Manipularea roboților în medii neprietenioase
- Filtrarea spam-urilor
- etc.

# Tipuri de IA

Google

amazon

IMDb

You Tube

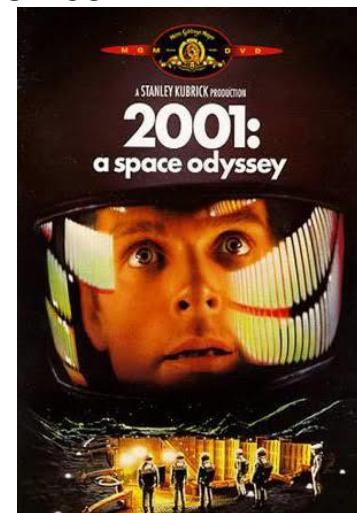
facebook

- **Narrow IA**

- Ceea ce vedem deja in calculatoare azi: sisteme inteligente care au fost invatate sa efectueze sarcini
- Exemple: optimizare in industrie, recunoastere voce (personal assistants), recunoastere imagini in self-driving cars, interpretare videos (drones), recomandare online pe baza unor preferinte, etc.

- **General IA**

- O forma flexibila de inteligenta capabila sa efectueze orice sarcina
- Survey (V. Muller, N. Bostrom, 2013):
  - 50% sanse sa fie dezvoltata in 2040-2050
  - → 90% in 2075
- Ceea ce vedem in filme

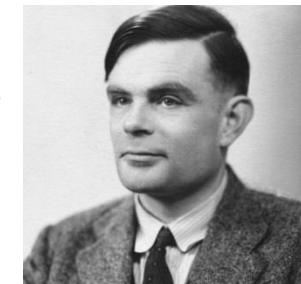


# Scurt istoric IA: etape majore

- Nașterea IA (1943-1956)
- Epoca de aur (1956-1974)
- Prima iarnă (1974-1980)
- Boom (1980-1987)
- A doua iarnă (1987-1993)
- IA meta-modernă (după 1993)

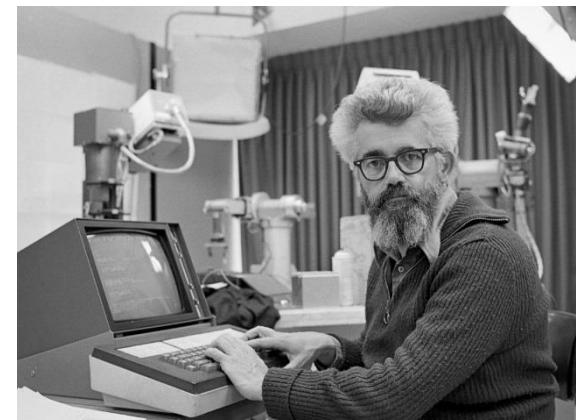
# Nasterea IA (1943-1956)

- Care sunt originile IA?
  - Matematică, logică, informatică, filozofie, psihologie, științe cognitive, biologie
- Primele noțiuni de IA
  - 1943 - Walter Pitts și Warren McCulloch propun neuronul artificial
  - 1950 - Alan Turing -> testul Turing
    - Mașinile pot gândi? Putem să ne dăm seama într-o conversație dacă interlocutorul este o mașină sau nu?
    - Demo - ALICE <http://www.alicebot.org>
  - 1951 - primele programe pentru jocuri (dame și șah)
  - 1955 - Allen Newell și Herbert Simon -> primul program pentru demonstrarea automată a teoremelor



# Nasterea IA (1943-1956)

- Conceptul de IA
  - 1956 - John McCarthy, școala de vară de la Dartmouth, SUA, propune termenul de IA
  - 1956 - John McCarthy face prima demonstrație cu rularea unui program de IA la CMU (Carnegie Mellon University)



“AI”

1950

1960

1970

1980

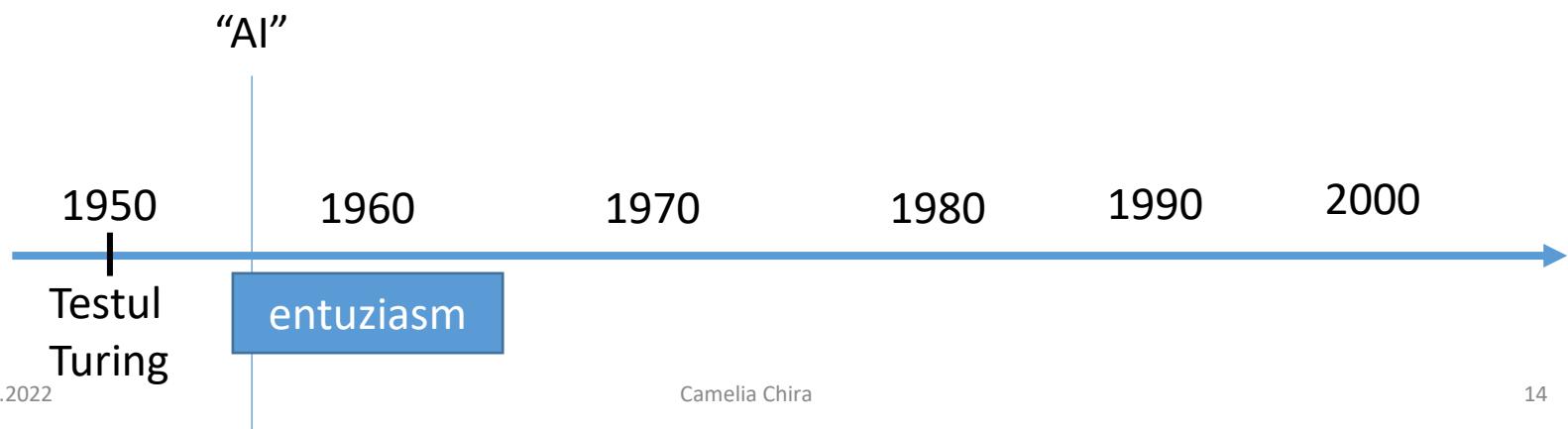
1990

2000

Testul  
Turing

# Epoca de aur (1956-1974)

- Calculatoarele pot executa o anumită sarcină X
  - X = rezolvarea puzzle-urilor, demonstrarea teoremelor geometrice, jucarea jocului de dame
  - Multe dintre aceste probleme - toy problems
  - 1958 - John McCarthy propune limbajul LISP la MIT (Massachusetts Institute of Technology)
  - 1965 - ELIZA (MIT, J. Weizenbaum)
  - 1969 - robotul Shakey combină locomoția, percepția și rezolvarea problemelor (Stanford Research Institute)
  - 1970 - “nașterea” algoritmilor evolutivi

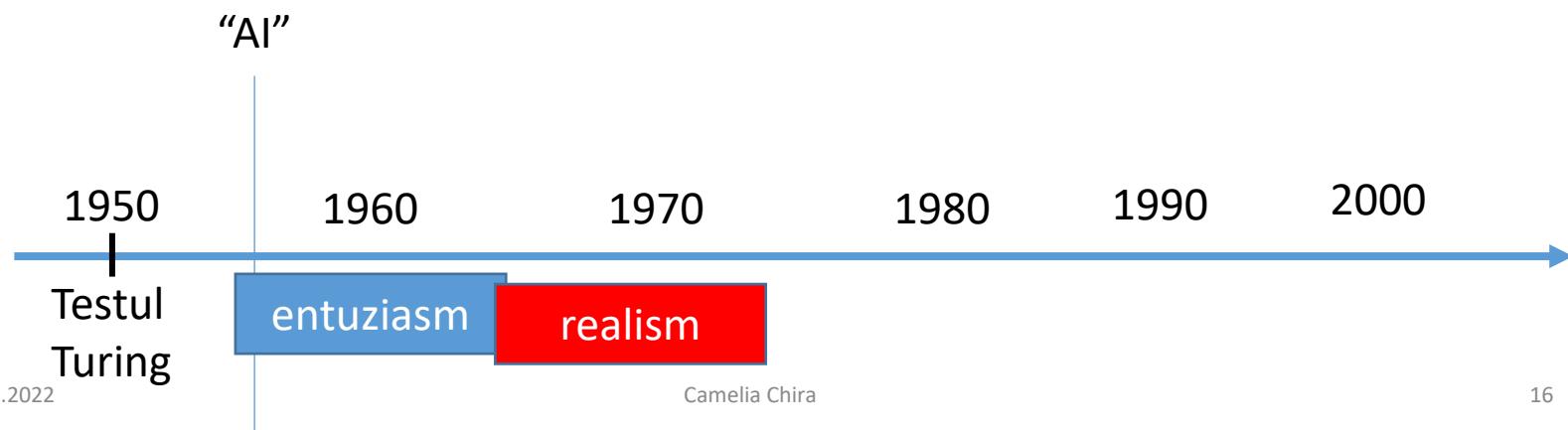


# Example: conversation with Eliza

```
Welcome to:  
      EEEEEE  LL      IIII  ZZZZZZ  AAAAAA  
      EE      LL      II      ZZ  AA  AA  
      EEEEEE  LL      II      ZZZ  AAAAAAAA  
      EE      LL      II      ZZ  AA  AA  
      EEEEEE  LLLLLL  IIII  ZZZZZZZZ  AA  AA  
  
Eliza is a mock Rogerian psychotherapist.  
The original program was described by Joseph Weizenbaum in 1966.  
This implementation by Norbert Landsteiner 2005.  
  
ELIZA: Is something troubling you ?  
YOU: Men are all alike.  
ELIZA: What is the connection, do you suppose ?  
YOU: They're always bugging us about something or other.  
ELIZA: Can you think of a specific example ?  
YOU: Well, my boyfriend made me come here.  
ELIZA: Is it important to you that your boyfriend made you come here ?  
YOU: He says I'm depressed much of the time.  
ELIZA: I am sorry to hear that you are depressed.  
YOU: It's true. I am unhappy.  
ELIZA: Can you explain what made you unhappy ?  
YOU: █
```

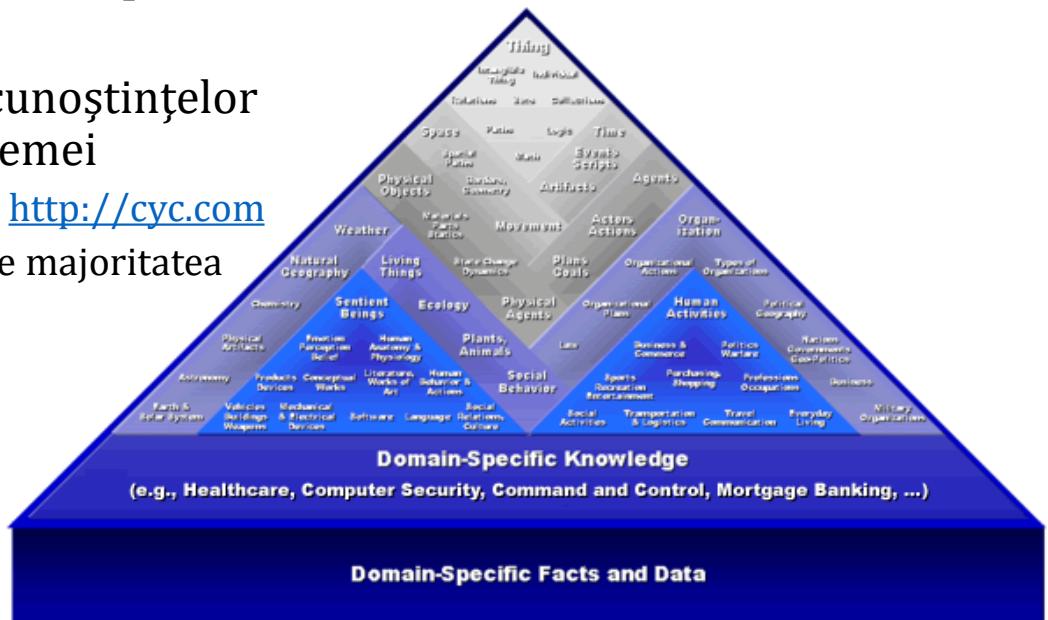
# Epoca de aur (1956-1974)

- 1966 – 1973 -> o doză de realism
  - Necesitatea cunoștințelor din domeniul problemei
    - Abordarea sintactică nu este suficientă -> translatorul automat rusă- engleză (US suspendă finanțarea)
  - Control dificil -> complexitate exponențială
    - Guvernul britanic suspendă finanțarea IA -> raportul lui Lighthill -> opinie pesimistă asupra cercetării în domeniul IA
  - Limite teoretice -> perceptronul nu poate rezolva problema XOR-ului
    - cercetarea rețelelor neuronale este suspendată



# Epoca de aur (1956-1974)

- 1969 – 1988 -> sisteme bazate pe cunoștințe
    - Ghidarea căutării pe baza cunoștințelor specifice domeniului problemei
      - Cyc -> o bază de cunoștințe- <http://cyc.com>
      - Sisteme expert dezvoltate de majoritatea companiilor



“AI”

1950

1960

1970

1980

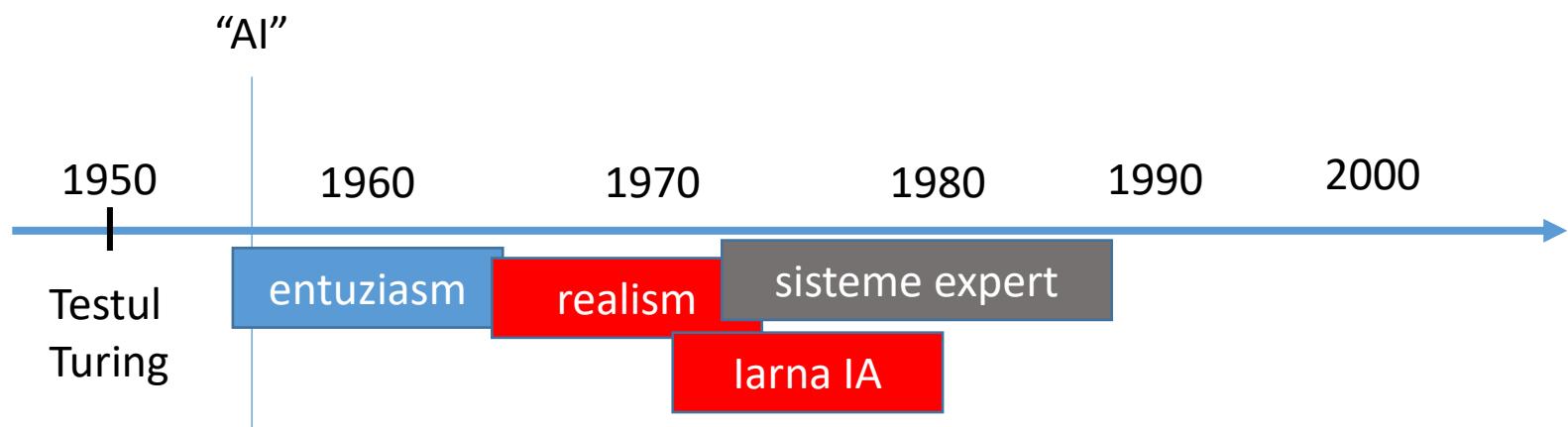
1990

2000

# Testul Turing

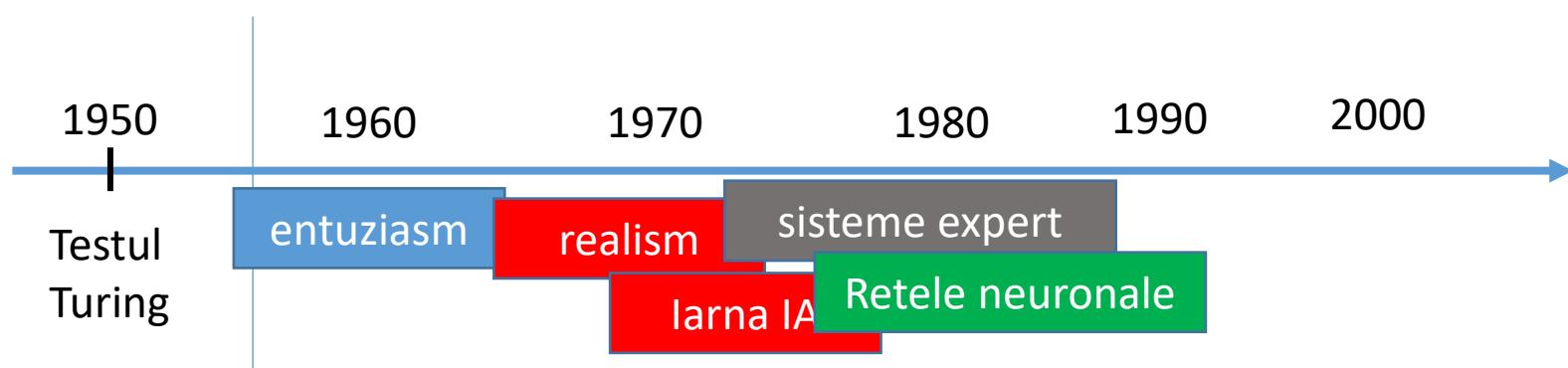
# Prima iarnă (1974-1980)

- Probleme
  - Puterea limitată a calculatoarelor
  - Creșterea exponențială a timpului necesar rezolvării unei probleme cu tehnici IA
  - Necesitatea unei baze de cunoștințe specifice domeniului problemei
  - Sistarea finanțării



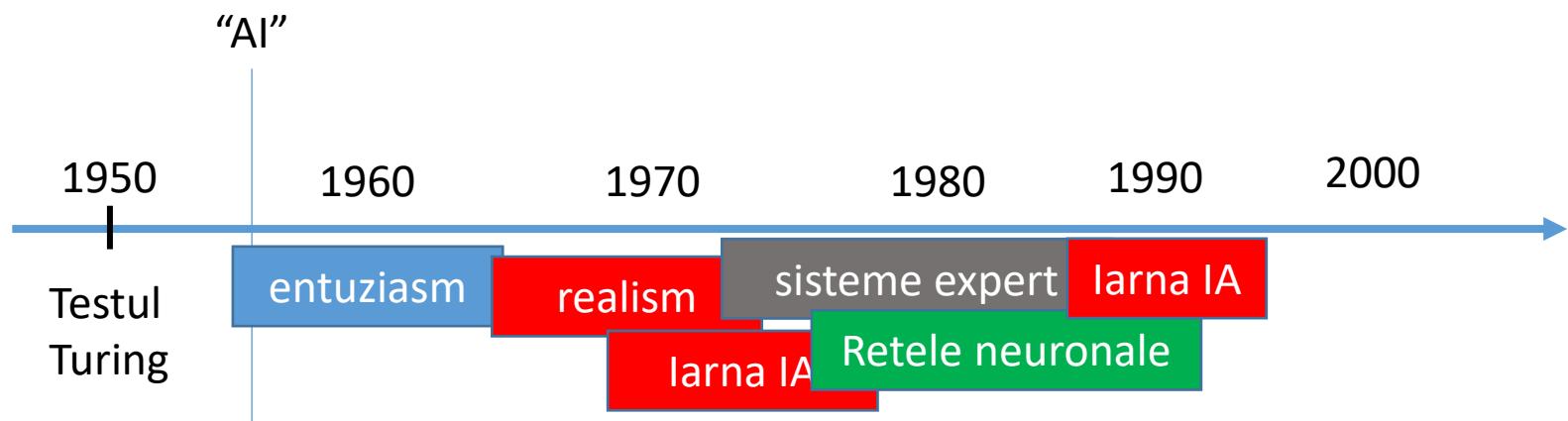
# Boom (1980-1987)

- Se dezvoltă puternic sistemele expert
  - MYCIN – Standford University -> diagnosticul bolilor infecțioase de sânge
  - XCON (eXpert CONfigurer) - Carnegie Mellon University -> Selectarea componentelor unui calculator în funcție de opțiunile utilizatorului
- 1986 – rețele neuronale artificiale
  - Perceptronul multistrat
  - Redescoperirea algoritmului de antrenare backpropagation
  - Noi dezvoltări:
    - Modelele simbolice (Newell, Simon)
    - Modelele logistice (McMarthy)



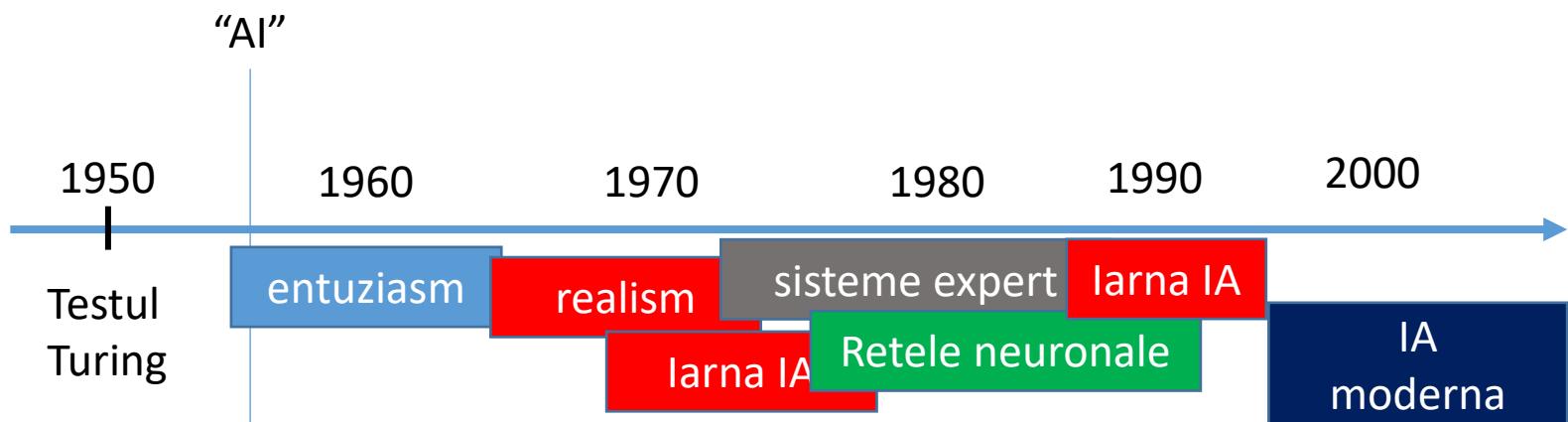
# A doua iarnă (1987-1993)

- Puterea de calcul limitată
- Suspiciunea companiilor
  - Banii au fost dirijați spre alte domenii de cercetare (diferite de IA)



# IA meta-modernă (1993 – prezent)

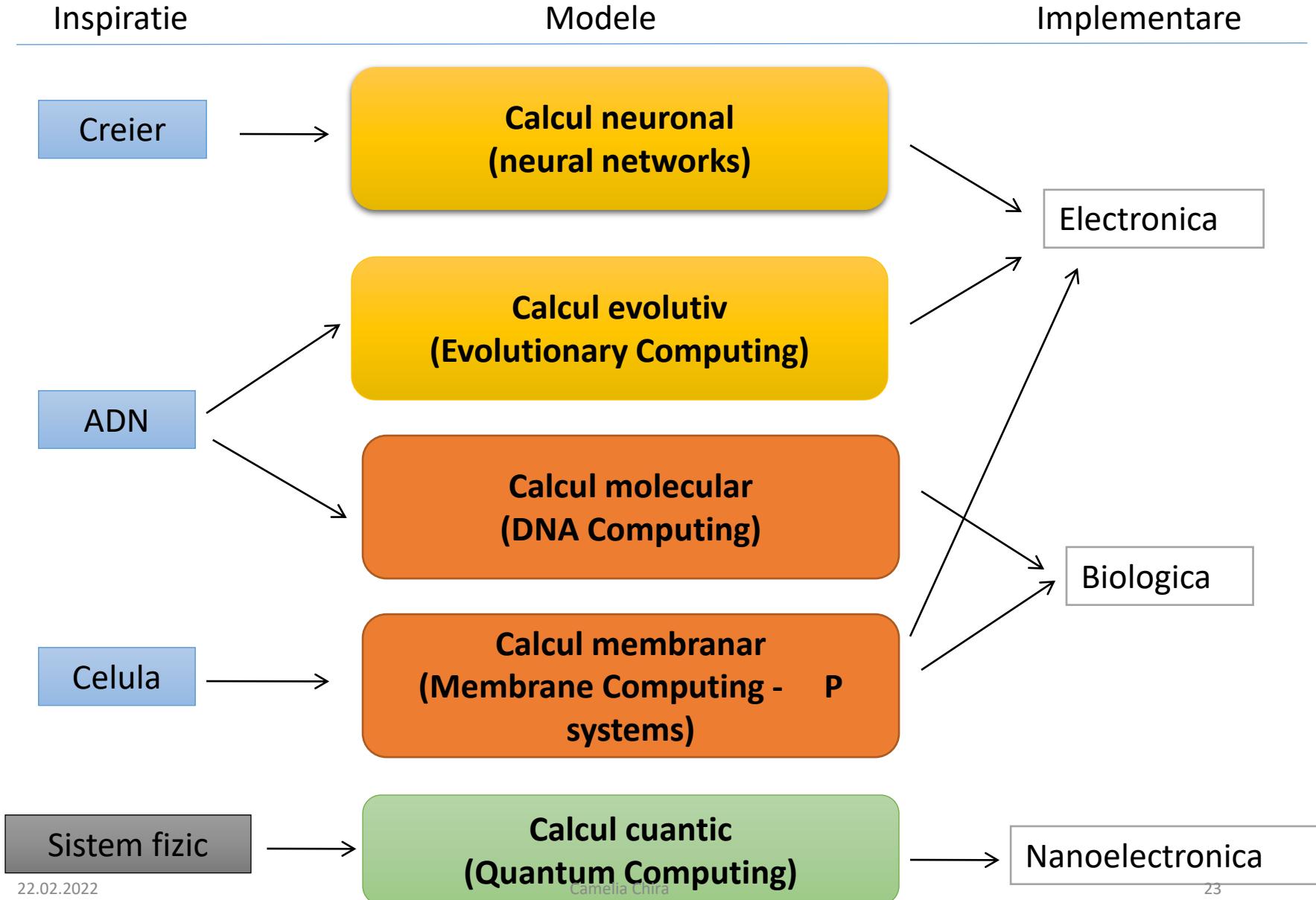
- 1997 – Deep Blue
- 1998 – programarea genetică
- 2000 – roboți pe post de animale de companie



# Inteligenta Artificială

- Artificial Intelligence (sometimes Computational Intelligence sau Soft Computing)
- **Calcul evolutiv** – algoritmi evolutivi
- **Calcul neuronal** – retele neuronale
- **Calcul fuzzy** – sisteme fuzzy
- Inteligența de grup – **swarm intelligence**
- Invățare automată – **machine learning**

# Calcul Natural

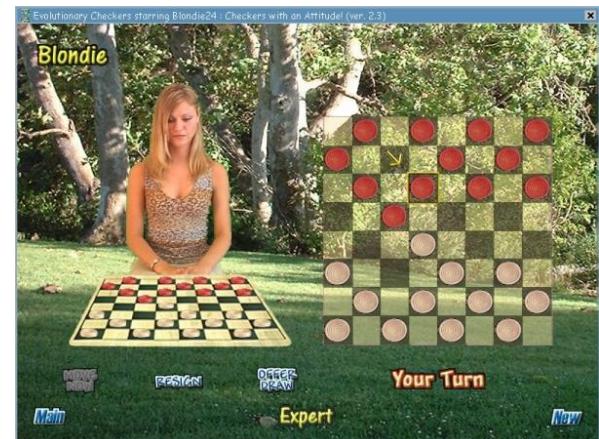


# Paradigma IA folosita cu succes:

- Impact social si economic
- Exemple
  - Folosirea motoarelor de cautare online
  - Sistemul bancar: citirea cecurilor prin coduri zip sau citirea cecurilor scrise de mana
  - Obtinerea directiilor de condus pe o anumita ruta
  - Algoritmi de detectare a fraudelor
  - Filtru spam
- Boeing, NASA – algoritmi genetici pentru optimizari
- 2011: Google self-driving cars
  - [https://www.youtube.com/watch?v=eXeUu\\_Y6W0w](https://www.youtube.com/watch?v=eXeUu_Y6W0w)

# Aplicatii in jocuri

- IBM – Deep Blue
  - A castigat in partida de sah cu Garry Kasparov, 1997
- Blondie24
  - A castigat un turneu de dame
  - A jucat impotriva a 165 oameni cu o rata mai buna decat ~99%
- Multe jocuri comerciale folosesc tehnici AI



# Aplicatii in jocuri

- **2011:** IBM's Watson a invins 2 castigatori anteriori ai jocului televizat Jeopardy
  - Sistemul a fost dezvoltat pentru a raspunde intrebarilor in limbaj natural
- **2015:** AI system called Claudico was 4<sup>th</sup> on a poker game
  - Poker: informatii lipsa!

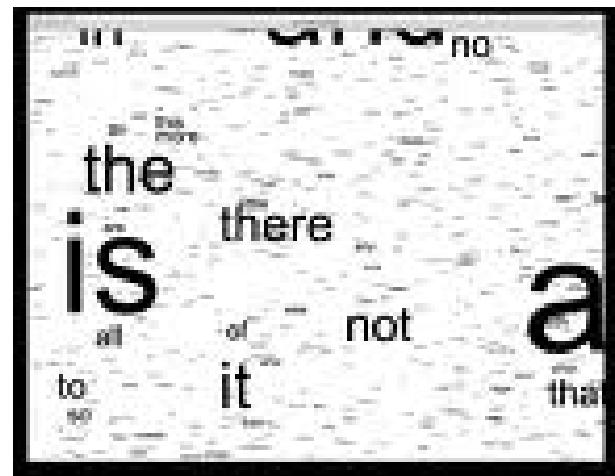


# Aplicatii in Logistica

- Itinerarii de drum
  - Directii de drum de la o locatie data la destinatie
  - Exemplu: MapQuest
- Analiza dinamica si replanificare
  - DART (Dynamic Analysis and Replanning Tool)
  - 1991 Razboiul din Golf, admin peste 50000 oameni, vehicule si echipamente cargo
- Planificarea traficului aerian
  - Mai ales in cazul intarzierilor sau re-rutarilor
  - Replanificare plecari/sosiri

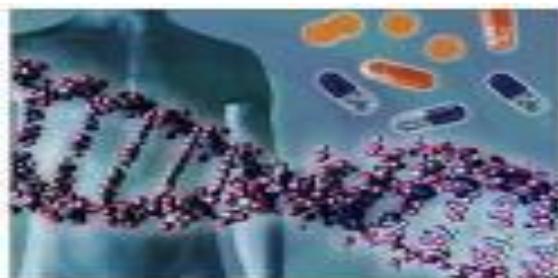
# Aplicatii in procesarea de text

- Traducerea automata dintr-o limba in alta
- Descoperirea informatiilor
- Clasificarea si organizarea textului
- Prezentarea unor informatii summarizate



# Alte Aplicatii

- Recunoasterea vorbirii
  - Sisteme de rezervari
  - Transcrierea automata - monitorizarea continutului in programele live de radio si televiziune
- Recunoasterea formelor
  - Recunoasterea scrisului de mana
  - Recunoasterea fetei
- Biologie si medicina
  - Diagnoza automata
  - Analiza genomului



# Probleme

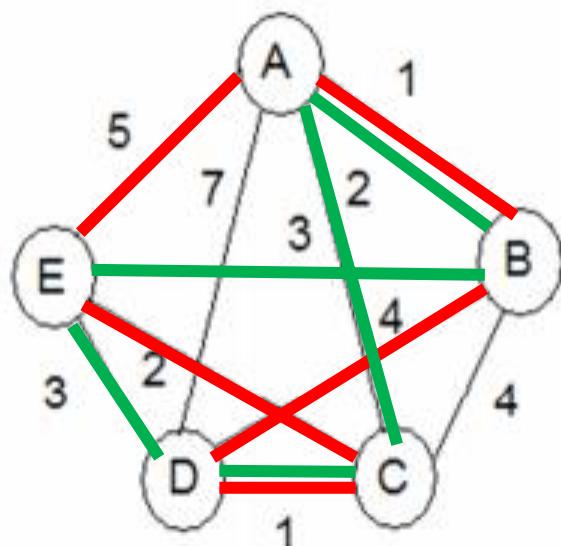
- O **problemă** este descrisa de mai multi *parametri* si *cerinte* pe care o solutie a problemei trebuie sa le satisfaca
- O **instanta a problemei** este obtinuta prin specificarea unor valori particulare pentru parametrii problemei
- Un algoritm este o procedura de rezolvare a problemei.
- Un algoritm rezolva o problema  $P$  daca poate fi aplicat oricarei instante  $I$  a lui  $P$  si garanteaza intotdeauna obtinerea unei solutii pentru instanta  $I$



# Problema comis-voiajorului

*Comis-voiajorul trebuie sa viziteze fiecare oras si sa se intoarca acasa pe drumul cel mai scurt.*

Fie  $G = (V, E)$  un graf neorientat în care oricare două vârfuri diferite ale grafului sunt unite printr-o latură căreia ii este asociat un cost pozitiv. Cerința este de a determina un ciclu care începe de la un nod, trece exact o dată prin toate celelalte noduri și se întoarce la nodul inițial, cu condiția ca ciclul să aibă un cost minim.

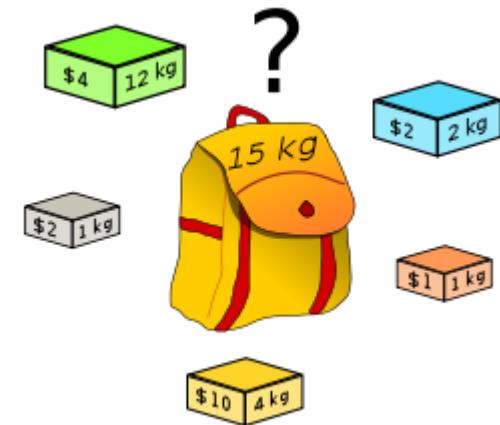


$$\text{A-B-D-C-E-A} = 1+4+1+2+5=13$$

$$\text{A-C-D-E-B-A} = 2+1+3+3+1=10$$

# Problema rucsacului (0/1 knapsack problem)

- Fiecare item are o valoare ( $v$ ) și o greutate ( $w$ ).
- Puneti în rucsac valoarea maximă fără a depasi greutatea maximă admisă.

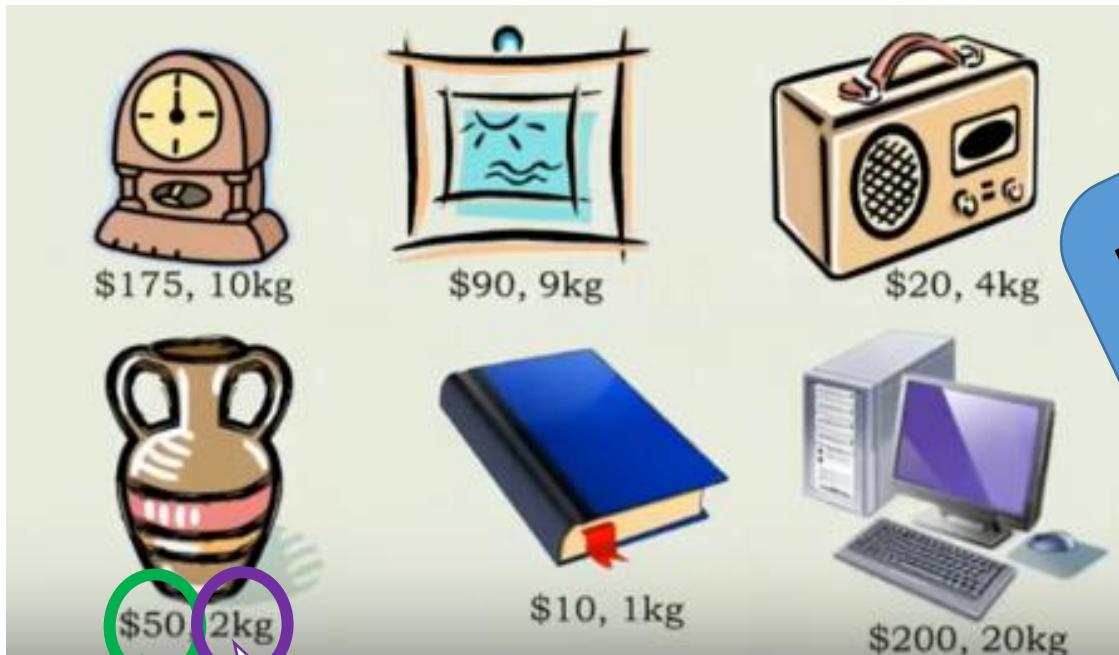


$$\begin{aligned} & \text{maximize} \sum_{i=1}^n v_i x_i \\ & \text{subject to} \sum_{i=1}^n w_i x_i \leq W \text{ and } x_i \in \{0, 1\}. \end{aligned}$$

# Exemplu: Burgler's dilemma



- Greutatea maxima a rucsacului : 20 kg



valoare

greutate

Problema de optimizare:  
Max. suma valorilor  
DAR  
Nu depasi greutatea  
totala de 20 kg!

0/1 knapsack problem

# Greedy approach



- Selecteaza cel mai bun item daca adaugarea lui in rucsac nu conduce la depasirea limitei de 20 kg
- *Cel mai bun?*
- **O metrica:**
  - *Max. valoare*
  - *Min. greutate*
  - *Max. valoare/greutate*
- Repeta pana cand rucsacul este plin
- Care metrica este cea mai buna?

# Cursul urmator

- Complexitate
- Probleme de optimizare
- Cautare locală



**BABEŞ-BOLYAI UNIVERSITY**  
Faculty of Mathematics and Computer Science



# Inteligentă Artificială

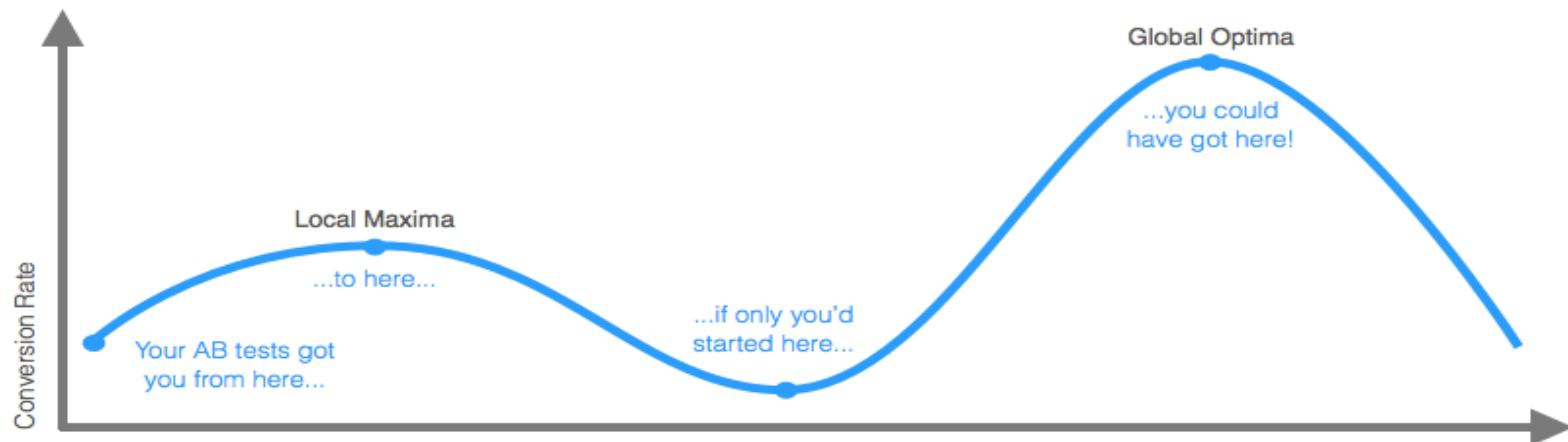
*2: Optimizare, cautare locală*

**Camelia Chira**

[camelia.chira@ubbcluj.ro](mailto:camelia.chira@ubbcluj.ro)

# Probleme de optimizare

- Scopul este de a determina minimul sau maximul unei anumite functii cu una sau mai multe variabile
- *Minimizare: Gaseste  $x^*$  in  $A$  astfel incat  $f(x^*) \leq f(x)$  pentru orice  $x$  in  $A$*
- Optim global, local



# Probleme de optimizare din lumea reală



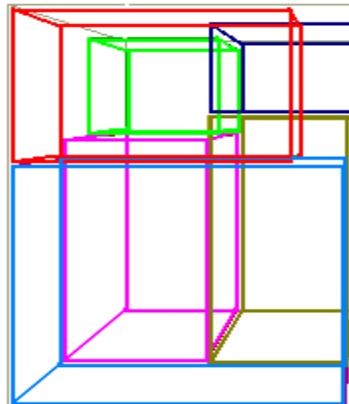
Logistica, transport



Planificare



Linii de producție, industrie  
01.03.2022



Ambalare,  
Camelia Chira  
distributie



Retele, telecomunicatii

# Optimizare

- Minimizare /maximizare
- Multe aplicatii
- Exemple
  - Gasirea de modele bool pentru propozitii
  - Gasirea celui mai scurt drum
  - Gasirea min/max pentru o functie complexa
  - Detectarea structurii comunitatilor in retele complexe
  - Determinarea structurii proteinelor
- Metode pentru probleme de optimizare
  - **Euristici si metaeuristică**
  - **Tehnici de cautare de tip single-point**
  - **Algoritmi evolutivi (bazati pe populatii)**

SAT

TSP

NLP

# The Boolean Satisfiability Problem (SAT)

- O expresie logica trebuie evaluata la TRUE
- Exemplu:
  - Gaseste valoarea fiecarui  $x_i$  pentru  $i=1,\dots,100$  astfel incat  $F(x)=\text{TRUE}$ 
$$F(x) = (x_{17} \vee \bar{x}_{37} \vee x_{73}) \wedge (\bar{x}_{11} \vee \bar{x}_{56}) \wedge \dots \wedge (x_2 \vee x_{43} \vee \bar{x}_{77} \vee \bar{x}_{89} \vee \bar{x}_{97})$$
  - $2^{100}$  posibilitati (2 valori pt fiecare din cei 100  $x_i$ )
- Marimea spatiului de cautare

$$|S| = 2^{100} \approx 10^{30}$$

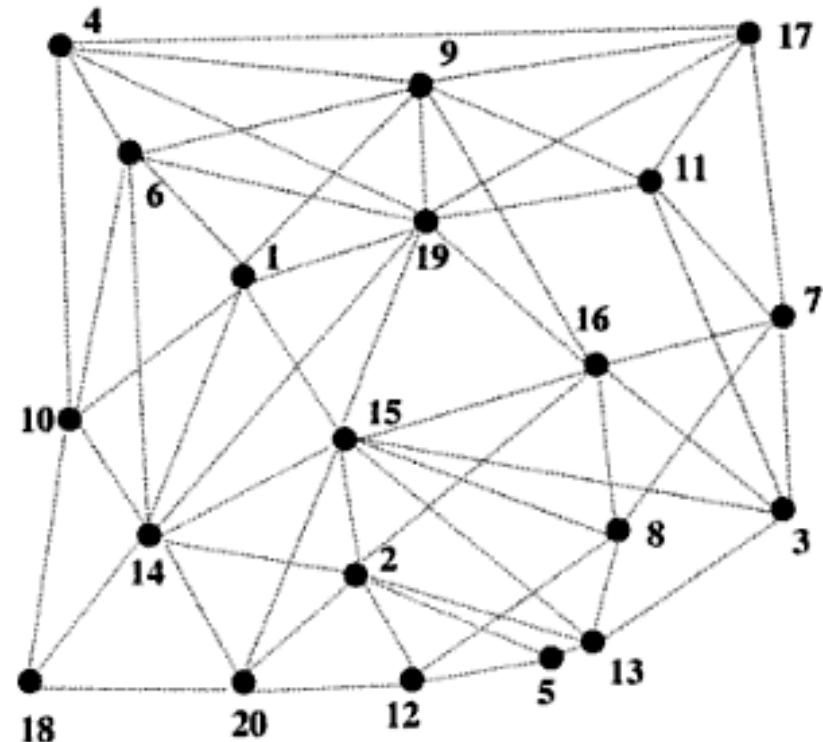
HUGE!!

Daca un calculator ar putea testa 1000 siruri/secunda si am incepe calculele la incepiturile timpului - acum 15 bilioane de ani- am ajunge sa examinam numai 1% din cazuri pana azi!!!!

# Traveling Salesman Problem (TSP)

## Problema comis-voiajorului

- Parcurgerea traseului pe cel mai scurt drum vizitand fiecare oras o singura data
  - Cost minim: timp, combustibil
- $\text{dist}(i,j) = \text{dist}(j,i)$ 
  - Altfel, TSP asimetrica
- Solutie: orice permutare de  $n$  orase



# TSP – spatiul de cautare

- $n!$  moduri de a permuta  $n$  orase
- Pentru  $n > 6$  numarul solutiilor posibile pentru TSP cu  $n$  orase este MAI MARE decat pentru SAT cu  $n$  variabile

$$|S| = n!/(2n) = (n-1)!/2$$

HUGE!!

n	TSP	SAT
6	60	64
7	360	128
8	2520	256

Nr orase	Nr solutii
10	181,000
20	10,000,000,000,000,000
50	100,000,000,000,000, 000,000,000,000, 000,000,000,000, 000,000,000,000,

# Nonlinear Programming Problem (NLP)

- Problema generala de optimizare: selecteaza n variabile  $x_1, x_2, \dots, x_n$  din domenii fezabile date astfel incat sa fie optimizata o functie  $f(x_1, x_2, \dots, x_n)$  si unele restrictii specificate sa fie respectate
- Ex. Maximizeaza functia:

$$G2(\mathbf{x}) = \left| \frac{\sum_{i=1}^n \cos^4(x_i) - 2 \prod_{i=1}^n \cos^2(x_i)}{\sqrt{\sum_{i=1}^n i x_i^2}} \right|,$$

subject to

$$\prod_{i=1}^n x_i \geq 0.75, \quad \sum_{i=1}^n x_i \leq 7.5n, \text{ and bounds } 0 \leq x_i \leq 10 \text{ for } 1 \leq i \leq n.$$

# NLP: maximizare

Maximizeaza  $f(x)$ ,  $x = (x_1, x_2, \dots, x_n) \in \mathbb{R}^n$

$$x \in F \subseteq S, \quad S \subseteq \mathbb{R}^n$$

$$l_i \leq x_i \leq u_i, 1 \leq i \leq n$$

F este definit prin  $m$  restrictii:

$$\begin{aligned} g_j(x) &\leq 0, j = 1, \dots, q \\ h_j(x) &= 0, j = q + 1, \dots, m \end{aligned}$$

- Spatiul de cautare este n-dimensional
- Nici o metoda de optimizare traditionala nu a dat rezultate satisfacatoare

# NLP – spatiul de cautare

- Depinde de numarul de dimensiuni
- Pur matematic: fiecare dimensiune poate contine un numar INFINIT de valori posibile

Pentru implementarea unui algoritm pe calculator:

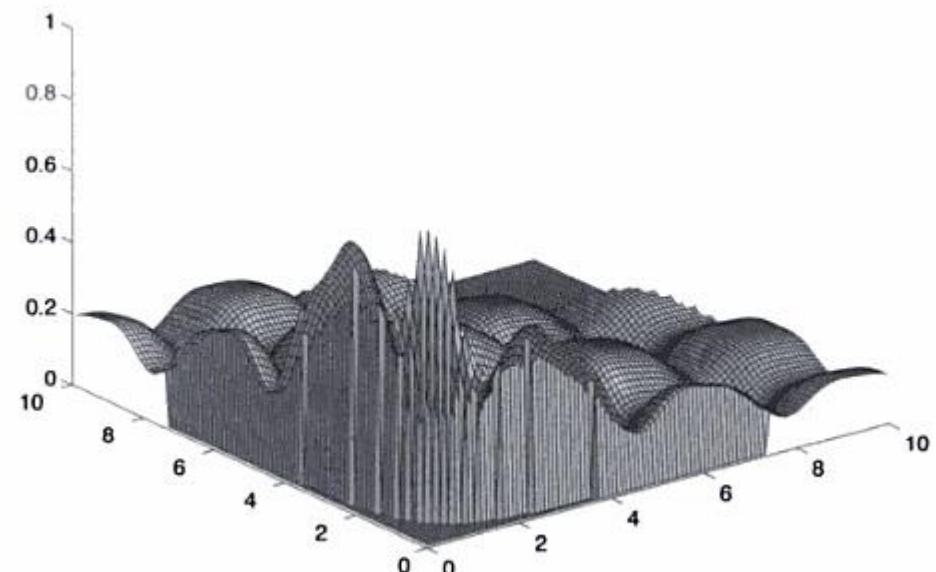
-precizia disponibila?

-pentru precizie de 6 puncte zecimali garantata fiecare variabila poate lua 10,000,000 valori

$$|S| = 10,000,000^n = 10^{7n}$$

HUGE!!

Chiar mai mare decat spatiul de cautare pentru TSP



# Modelarea problemei

- Modele – simplificarea lumii reale
- SAT, TSP, NLP – forme canonice de modele ce pot fi aplicate în multe situații

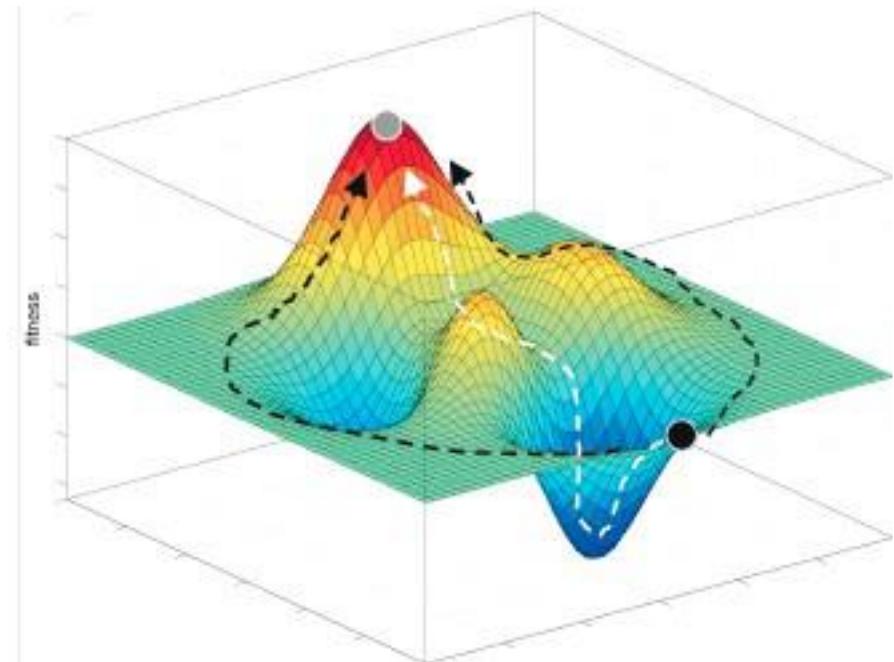


# De ce sunt unele probleme dificile?

- Numarul solutiilor posibile din spatiul de cautare este prea mare pentru a permite o cautare exhaustiva
- Problema este atat de complicata incat este necesara simplificarea ei pentru a obtine orice raspuns incat orice rezultat devine nefolositor
- Functia de evaluare a calitatii solutiilor gasite se schimba in timp (noisy) – nevoie de o serie de solutii si nu una singura
- Solutiile posibile sunt atat de restrictionate incat chiar si construirea uneia valide este dificila, una optima?

# Spatiul de cautare

- Spatiul de cautare este multimea solutiilor candidat ale problemei



## *Marimea problemei*

Exemplu. TSP cu  $n=10$  orase, marimea problemei este 10

## *Marimea spatiului de cautare*

Exemplu: TSP cu 10 orase are aprox. 181,000 solutii posibile

$$O(2^n)$$

$$O(n!)$$

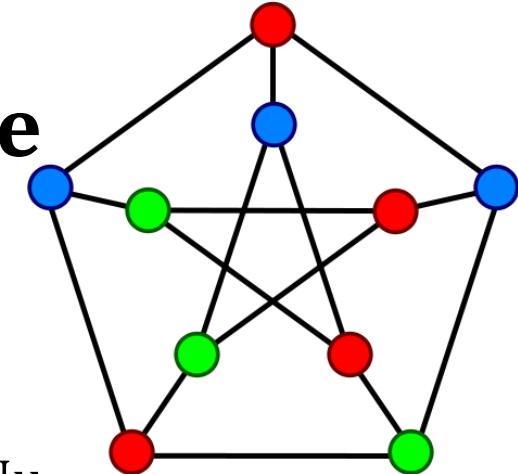


# Functia de evaluare

- Aceasta functie trebuie sa determine cat de buna este o solutie
- De obicei mapeaza o solutie la o valoare reala (care trebuie minimizata/maximizata)
- Poate fi si o functie de sortare a solutiilor: pentru orice pereche de solutii, determina care este mai buna
- Se mai numeste:
  - Functie obiectiv
  - Functia energie (fizica)
  - Functia de cost (economie)
  - Functia de calitate (inginerie)
  - Functia de fitness (biologie)

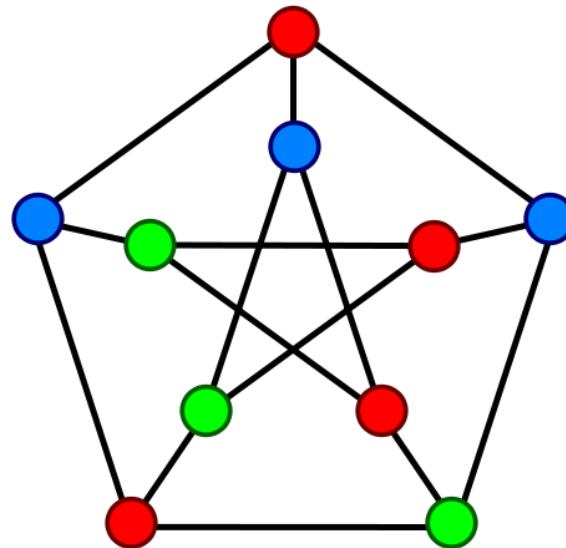
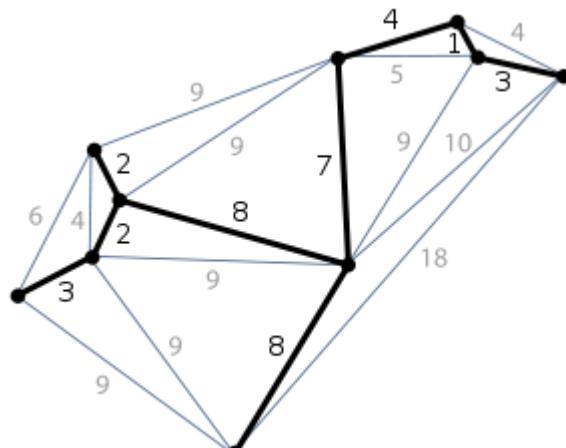
# Probleme: optimizare si decizie

- Problema vs. Instanta a unei probleme
  - O instantă poate avea mai multe soluții
- Probleme de decizie: cele care au un răspuns Da/Nu
  - Ex. Este un graf dat  $k$ -colorabil?
  - Dacă problema de optimizare poate fi rezolvată ușor atunci și problema de decizie corespunzătoare se rezolvă ușor
- O problema este **undecidable** dacă nu există nici un algoritm care primește ca input o instantă a problemei și determină răspunsul pentru acea instantă *ex. Turing's Halting problem*
- O problema este **tractabilă** dacă poate fi rezolvată de un algoritm în timp *polinomial*
- O problema este **intractabilă** dacă orice algoritm are nevoie de timp superpolinomial



# Clasa de complexitate P

- Multimea problemelor de decizie ce pot fi rezolvate in timp **polinomial** *i.e.  $O(n^k)$ , unde  $k$  este o constantă*
- Multe probleme sunt in clasa P *ex. minimum spanning tree*
- Nu toate problemele pot fi rezolvate in timp polinomial



# Clasa de complexitate NP (nondeterministic polynomial time)

- **NP** – probleme pentru care o solutie poate fi verificata in timp polinomial
- Probleme intractabile
  - Marimea problemei creste
  - nu mai pot fi rezolvate in timp rezonabil
- O problema de decizie este in clasa NP daca are un algoritm polinomial nondeterminist:
  - Algoritmul “ghiceste” o solutie (nondeterminist)
  - Verifica determinant in timp polinomial ca solutia este corecta

T(n)	Name	Problems
O(1)	Constant	Easy-solved
O(log n)	Logarithmic	
O(n)	Linear	
O(n log n)	Linear-logarithmic	
O(n <sup>2</sup> )	Quadratic	
O(n <sup>3</sup> )	Cubic	
O(2 <sup>n</sup> )	Exponential	Hard-solved
O(n!)	Factorial	

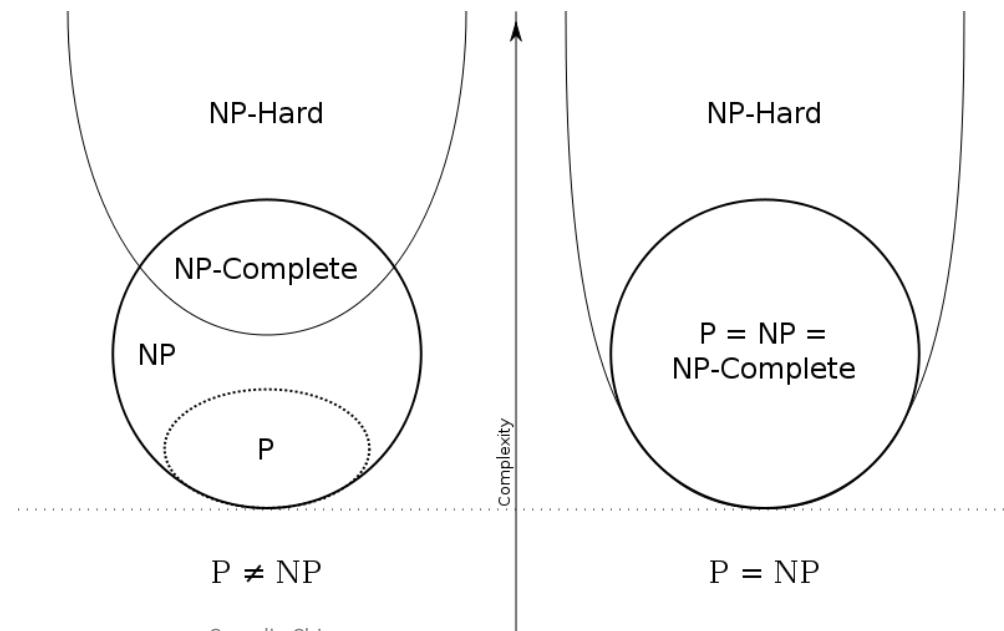
P=NP ?

# Probleme NP-complete

- *Toate problemele X din NP pentru care este posibil sa reducem orice alta problema Y din NP la X in timp polinomial*
- **Reducibility:** O problema P poate fi redusa la alta problema Q daca orice instanta a lui P poate fi transformata intr-o instanta a lui Q a carei solutie ofera o solutie instantei lui P
  - Daca P se reduce la Q, P nu se rezolva mai greu decat Q
- O problema este **NP-complete** daca:
  - $X \in NP$
  - $\forall Y \in NP, Y$  este reductibila la X in timp polinomial
- Probleme NP-complete:
  - Intractabile
  - Nu pot fi rezolvate in timp polinomial de nici un algoritm
  - Daca o problema NP-complete ar putea fi rezolvata in timp polinomial atunci orice problema NP-complete are un algoritm polinomial
  - Solutii aproximative
- **Ex. SAT**

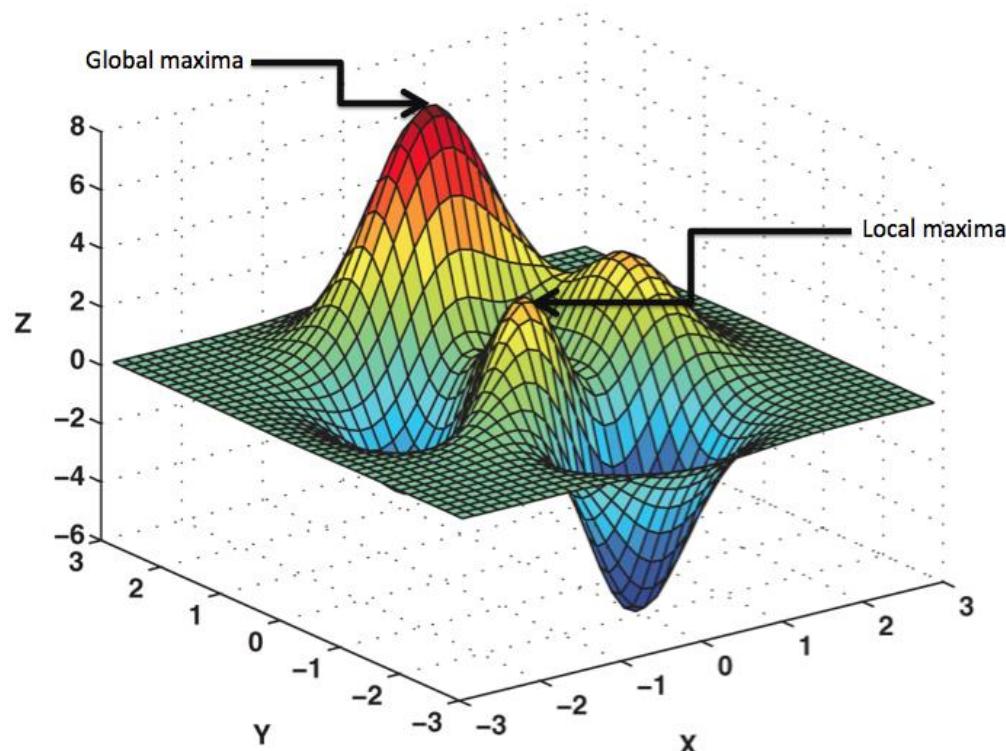
# Probleme NP-hard

- Probleme cel putin la fel de grele ca si problemele NP-complete
- O problema X este **NP-hard** daca exista o problema NP-complete Y astfel incat Y este reductibila la X in timp polinomial
  - Daca  $X \in NP$  si X este NP-hard atunci X este NP-complete
  - Daca X se poate reduce la Y si X este NP-complete atunci si Y este NP-complete
- Ex. TSP

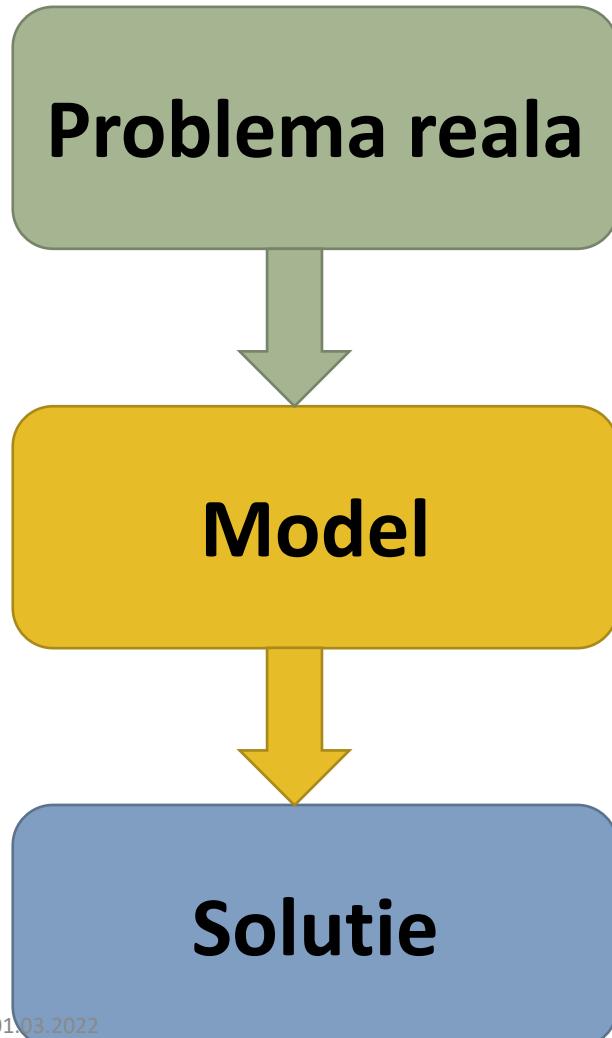


# Optimizare

- Într-o problema de optimizare, cautăm cea mai bună soluție care satisfăce una sau mai multe condiții
- Majoritatea problemelor de optimizare sunt în clasa de complexitate NP
- Spații de căutare complexe
- Global optima
- Local optima
- Multiple solutions



# Optimizare: probleme reale



## Model

- Reprezentarea problemei
- Restrictii
- Functia obiectiv

## Solutie a modelului

- O solutie candidat fezabila
- Conduce la o valoare optima sau aproape de optim a functiei obiectiv

## Algoritm de optimizare

- Exacti
- Euristică, metode aproximative

# Concepțe de bază

- Reprezentare
  - Cum poate fi reprezentată problema
  - Ce înseamnă o soluție potentială?
- Obiectiv
  - Care este obiectivul problemei?
  - Formulează matematic ce se cere
- Funcție de evaluare
  - Diferită de obiectiv!
  - Trebuie să returneze o anumita valoare care să indice calitatea unei soluții sau care să permită compararea a două soluții



# SAT: Reprezentare

- n variabile ce pot lua valoarea TRUE sau FALSE
- Sir de lungime n, cu valori posibile 0 sau 1
- Fiecare element din sir corespunde unei variabile
- Spatiul de cautare:  $|S| = 2^n$  (orice punct din S este o solutie fezabila)

## Exemplu

- F cu 8 clauze si 10 variabile

$$F = (x_1 \vee x_2 \vee \neg x_3) \wedge (x_2 \vee \neg x_{10}) \wedge (\neg x_2) \wedge (x_4 \vee x_{10}) \wedge (x_3 \vee x_5) \wedge (\neg x_4 \vee x_2 \vee \neg x_5) \wedge (\neg x_1 \vee x_6 \vee \neg x_7) \wedge (x_8 \vee x_{10})$$

- Exista  $2^{10} = 1024$  posibile solutii, dintre care numai 32 (3%) sunt valide
- O posibila solutie: un sir de 10 biti
- x=0111000110 inseamna  $x_1=0, x_2=1, x_3=1$ , etc.

# TSP: Reprezentare

## TSP cu n orase

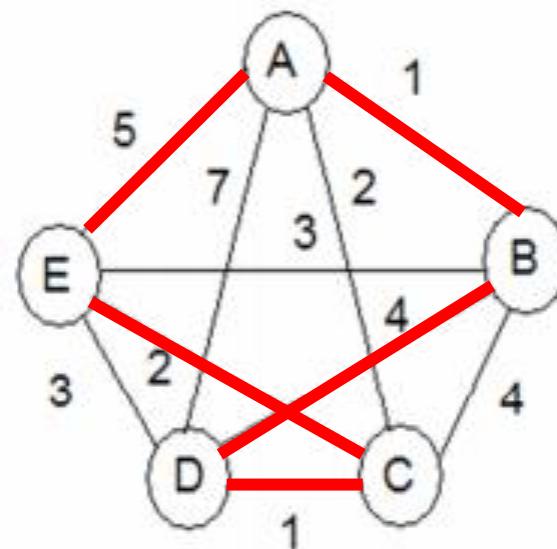
- Permutare de numere naturale 1...n
  - Fiecare numar ii corespunde unui oras
  - Ordinea de vizitare a oraselor este data de permutare
- Spatiul de cautare:  $|S| = (n - 1)! / 2$

Exemplu

5 orase: A,B,C,D,E

A-B-D-C-E-A

1-2-4-3-5-1



# NLP: Reprezentare

- Spatiul de cautare este aici format din numere reale in **n** dimensiuni
- Un n-tuplu de n numere reale (vector de numere reale)

$$x = (x_1, x_2, \dots, x_n) \in \mathbb{R}^n$$

- Marimea spatiului de cautare depinde de precizie
  - Pentru 6 pozitii zecimale si numere x cuprinse intre 0 si 10:

$$|S| = 10,000,000^n = 10^{7n}$$

# Reprezentare: observatii

- Pentru fiecare problema, reprezentarea unei solutii si interpretarea ei conduce la **spatiul de cautare** si dimensiunea lui
- Spatiul de cautare nu este determinat de problema!
  - De reprezentare si felul in care solutiile sunt codificate

# Obiectivul problemei

- Ce cautam? Care este obiectivul problemei date?
- Mai degraba o expresie decat o functie

- **SAT**

gasirea unui vector de biti astfel incat functia F sa fie evaluata la TRUE

- **TSP**

minimizarea distantei totale parcurse de comis-voiajor cu restrictia de vizita fiecare oras o singura data si de a ajunge inapoi la orasul initial

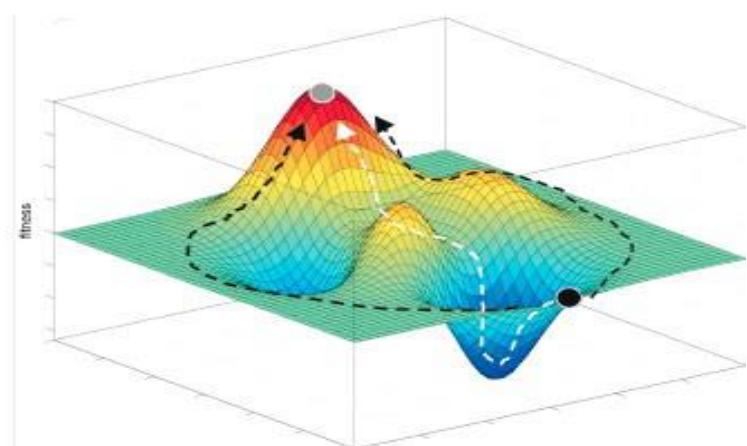
- **NLP**

minimize or maximize of functie neliniara care primeste solutiile candidat

# Functia de evaluare

- De obicei mapeaza spatiul solutiilor posibile sub reprezentarea aleasa la un set de numere (e.g. reale)
- Fiecare element din spatiul de cautare primeste o valoare numerica care indica calitatea acelei solutii
- Compararea solutiilor (*ordinal eval functions*) vs calitatea solutiilor (*numeric evaluation functions*)
- Cum setam functia de evaluare?
  - Criteriu evident: O solutie care satisface obiectivele problemei in intregime ar trebui sa aiba cea mai buna evaluare.
  - De multe ori, obiectivul problemei sugereaza o functie de evaluare (e.g. TSP, NLP)
  - SAT?

# Problema de optimizare



- Problema de cautare
- Spatiu de cautare  $S$ 
  - Partea fezabila  $F$  a lui  $S$ : contine solutii ce satisfac toate restrictiile problemei
  - TSP, NLP:  $F=S$

## *Definitie (problema de minimizare)*

Dat fiind un spatiu de cautare  $S$  si partea lui fezabila  $F$ ,  $F \subseteq S$ , sa se gaseasca  $x \in F$  astfel incat  $eval(x) \leq eval(y), \forall y \in F$ .

Punctul  $x$  care satisface conditia de mai sus se numeste **solutie globala**.

# Cei trei pasi spre rezolvare...

- Cautare: o plimbare adaptata spatiului (fitness landscape)
- Orice problema de optimizare/cautare are 3 componente principale:

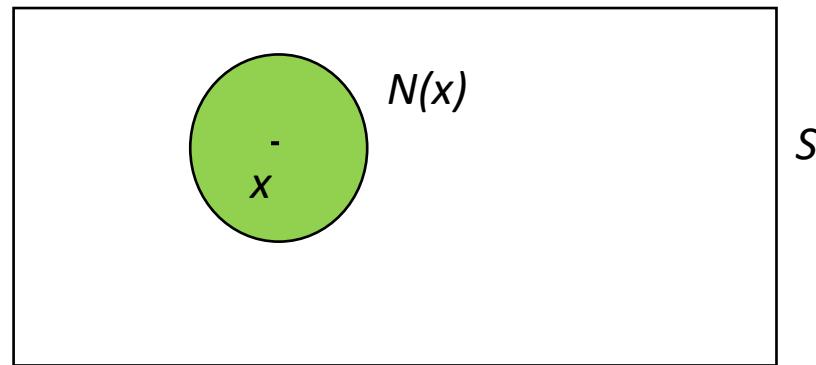
$$L = (S, f, d)$$



1. **Reprezentare** – multime de solutii posibile – *spatiu de cautare* (S)
2. **Functie de evaluare** (definite pornind de la obiectiv) – functia obiectiv – calitatea solutiilor
3. **Functia de vecinatate** – o metrica de distanta care sa determine solutii din vecinatate

# Vecinatate si optime locale

- Vecintatea unui punct  $x$  din  $S$  este multimea acelor puncte care sunt “aproape” de  $x$  intr-un mod masurabil.



- Cand sunt 2 puncte apropiate?

# Vecinatate: Cand sunt 2 puncte apropiate?

Putem defini o functie de distanta  $dist$  pe  $S$

$$dist: S \times S \rightarrow \mathbb{R}$$

si o vecinatate  $N(x)$  pentru un  $\varepsilon \geq 0$ :

$$N(x) = \{y \in S : dist(x, y) \leq \varepsilon\}$$

- Pentru spatii continue, e.g. NLP, vecinatatea poate fi definita folosind distanta euclidiana:

$$dist(x, y) = \sqrt{\sum_{i=1}^n (x_i - y_i)^2}$$

- SAT: distanta dintre 2 puncte poate fi **Hamming distance** (= numarul de pozitii care au valori diferite)

# Vecinatate: Cand sunt 2 puncte apropiate?

Putem defini o mapare  $m$  pe  $S$  care defineste o vecinatate pentru fiecare punct  $x$  din  $S$ :

$$m: S \rightarrow 2^S$$

## TSP: 2-swap mapping

O permutare de  $n$  orase are  $n(n-1)/2$  vecini

1-2-3-4 -> 1-3-2-4

Ex.  $n=20$ ,  $x= 15-3-11-19-17-2-\dots-6$

Vecinatate *2-swap* (190 puncte in  $S$ ):

**15-17-11-19-3-2-\dots-6**

**2-3-11-19-17-15-\dots-6**

**15-3-6-19-17-2-\dots-11**

## SAT: 1-flip mapping

(inversarea unui singur bit)

0->**1**    1->**0**

Un sir de  $n$  biti are  $n$  vecini.

Ex.  $n=20$ ,  $x=01101010001000011111$

Vecinatate *1-flip* (20 de puncte in  $S$ ):

**11101010001000011111**

**00101010001000011111**

**01001010001000011111**

# Optime locale

- O solutie potentiala este optim local in relatie cu vecinatatea  $N$  daca si numai daca

$$eval(x) \leq eval(y), \forall y \in N(x)$$

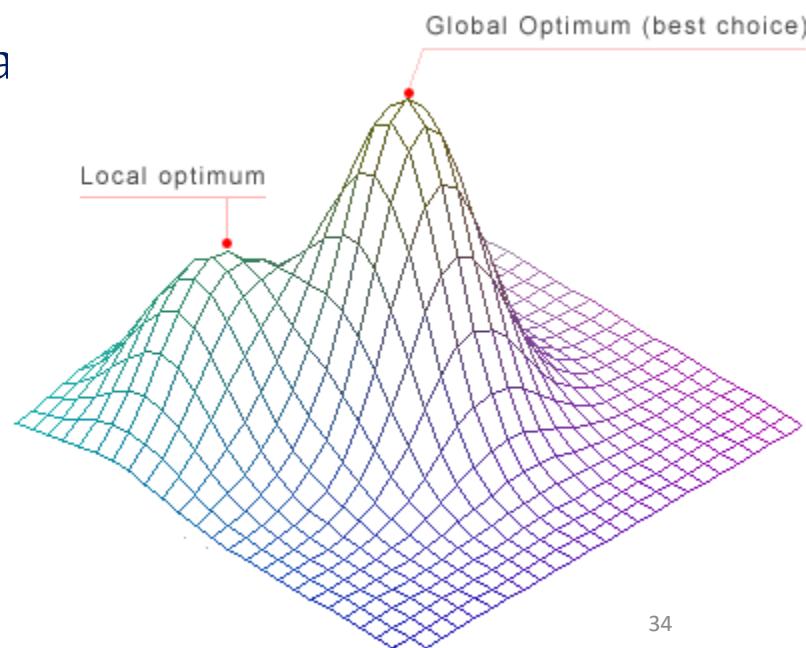
- Usor de verificat pentru vecinatati relativ mici

- Multi algoritmi se bazeaza pe statistica vecinatatii: solutiile generate se bazeaza numai pe *informatie locala* la fiecare pas (*local search strategies*)

- *Exemplu:*

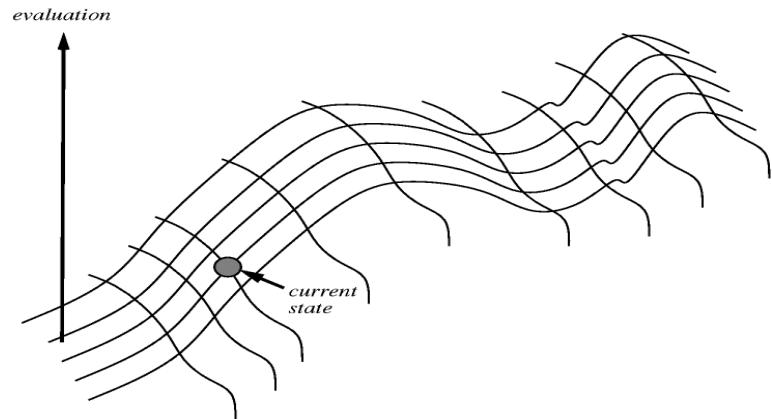
$$\max. f(x) = -x^2, \varepsilon = 0.1$$

(vecinatate  $+/- 0.1$  pentru un punct  $x$ )



# Cautare locală

- **Hill-climbing (HC):**  
imbunatatire iterativa
- Tehnica se aplica unui singur punct (punct curent) din spatiul de cautare
- La fiecare iteratie, un nou punct este selectat din vecinatatea punctului curent
- Daca noul punct are o evaluare mai buna (functia obiectiv!), atunci el devine punctul curent
- .... pana cand solutia nu se mai imbunatatesta sau max iteratii
- **Variante HC**
  - **Random HC**
  - **Steepest-ascent HC**
  - **Next-ascent HC**



# Random HC (RHC)

1. Se selecteaza un punct aleator  $c$  (*current*) in spatiul de cautare
2. Se alege un punct  $x$  din vecinatatea lui  $c$ :  $N(c)$ .  
Daca  $eval(x)$  este mai bun decat  $eval(c)$  atunci  $c=x$ .
3. Repeta pasul 2 pana cand un numar maxim de evaluari se atinge.
- 4. Returneaza  $c$ .**

# Steepest Ascent HC (SAHC)

1. Se selecteaza un punct aleator  $c$  (*current hilltop*) in spatiul de cautare.
2. Se determina toate punctele  $x$  din vecinatatea lui  $c$ :  $x \in N(c)$
3. Daca oricare  $x \in N(c)$  are un fitness mai bun decat  $c$  atunci  $c=x$ , unde  $x$  are cea mai buna valoare  $\text{eval}(x)$ .
4. Daca nici un punct  $x \in N(c)$  nu are un fitness mai bun decat  $c$ , se salveaza  $c$  si se trece la ***pasul 1***. Altfel, se trece la ***pasul 2*** cu noul  $c$ .
5. Dupa un numar maxim de evaluari, se returneaza cel mai bun  $c$  (hilltop).

a.k.a.

**Best-Improvement HC**

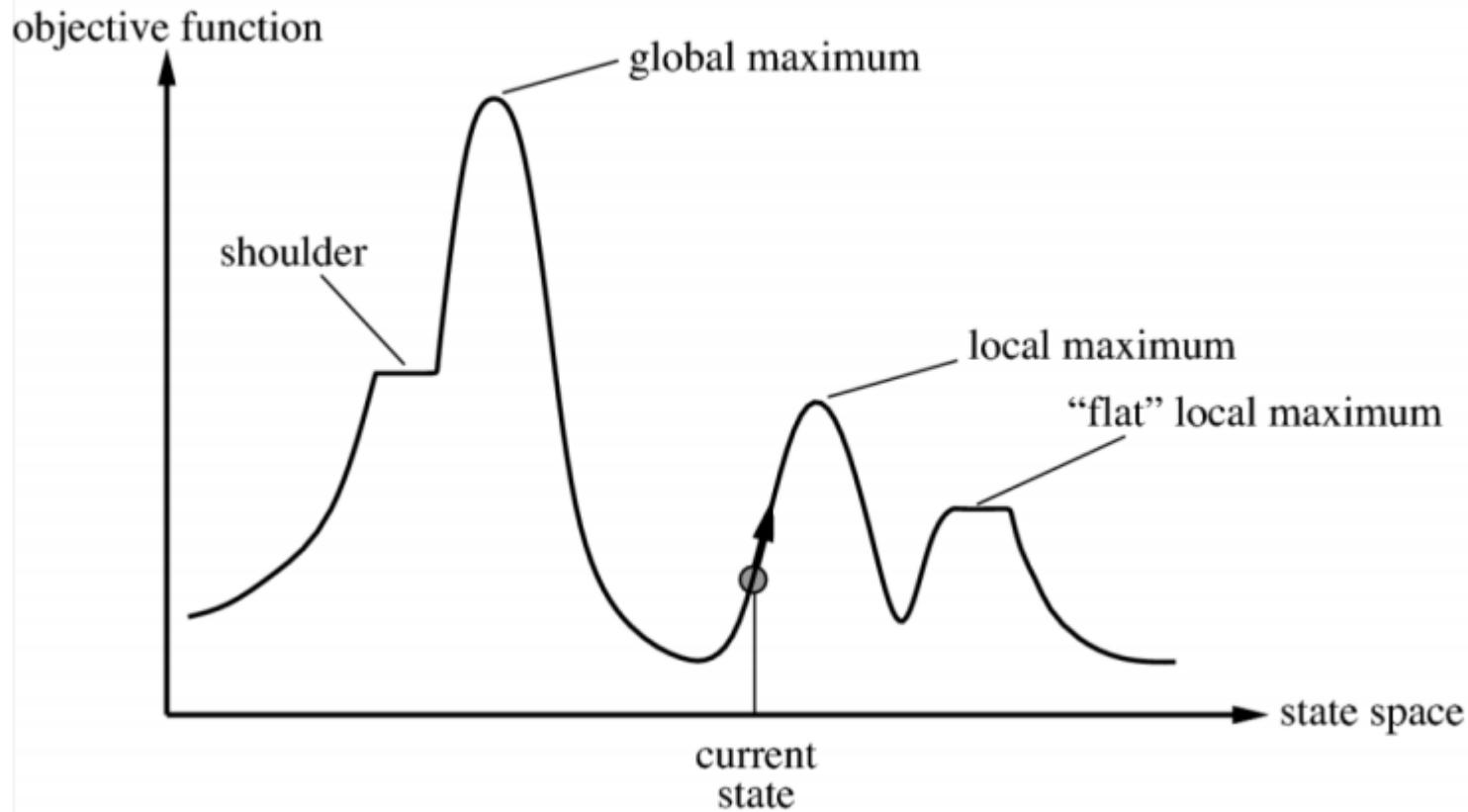
# Next Ascent HC (NAHC)

1. Se selecteaza un punct aleator  $c$  (*current hilltop*) in spatiul de cautare.
2. Se considera pe rand vecinii  $x$  ai punctului  $c$ . Daca  $\text{eval}(x)$  este mai bun decat  $\text{eval}(c)$ , atunci  $c=x$  si nu se mai evalueaza restul vecinilor lui  $c$ . Se continua pasul 2 cu noul  $c$  si se considera vecinii lui  $c$  mai departe (pornind din acelasi punct din vecinatate unde s-a ramas cu vechiul  $c$ ).
3. Daca nici un vecin  $x$  al punctului  $c$  nu duce la o evaluare mai buna, se salveaza  $c$  si se continua procesul de la pasul 1.
4. Dupa un numar maxim de evaluari, se returneaza cel mai bun  $c$  (*hilltop*).

*a.k.a.*

**First-Improvement HC**

# Hill-climbing si S



# Hill-climbing: dezavantaje

- De obicei, metodele HC se opresc in optime locale
- Nu se stie cat de mult optimele locale descoperite deviaza de la optimul global, sau chiar de la alte optime locale
- ***Optimul obtinut depinde de configuratia initiala***
- Nu este posibil sa specificam un timp maxim de calcul
- *Explorare - exploatare*

....DAR:

- ✓ **Usor de implementat si aplicat**
- ✓ **Rezultate bune ca si metode de cautare locala**

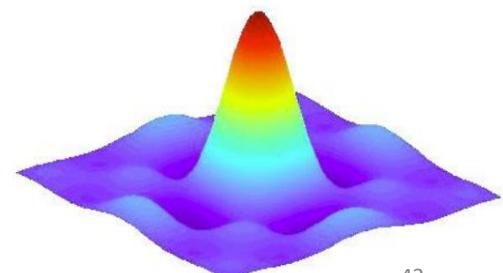


# Stuck in local optima?

- Nu! Exista metode de cautare inteligenta...
- Metoda aleasa depinde intotdeauna de problema
- Opsiuni?
  - Executa algoritmul pentru un numar mai mare de configuratii initiale
  - Foloseste rezultatele intermediare pentru a imbunatati alegerea punctului curent in iteratia urmatoare
  - Mareste vecinatatea
  - Modifica criteriul de acceptare a unor noi puncte astfel incat sa fie permise cateodata si puncte care au o evaluare mai proasta
- Metode?
  - Random restart HC
  - Simulated Annealing
  - Tabu search

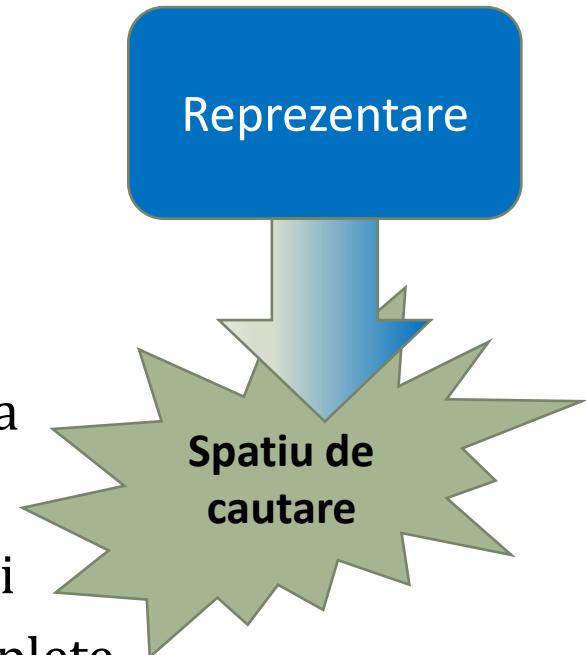
# Random Restart HC

- Cand se ajunge intr-un optim local, se alege un alt punct curent in mod aleator
- HC este rulat de mai multe ori incepand din alte puncte initiale
- Avantaje?
  - Mai multe incercari de a gasi o zona buna a spatiului de cautare
  - Costul computational nu este mai mare
- **Numarul de incercari creste => probabilitatea de a gasi solutia se apropie de 1**
  - Daca fiecare HC are probabilitatea de success  $p$ , numarul de restarturi necesare este aproximativ  $1/p$



# Metode clasice

- Spatiul de cautare determinat de o anumita reprezentare poate fi considerat ca si multimea solutiilor complete => cautare in acest spatiu
- Spatiul de cautare poate fi divizat in subspatii si cautarea sa vizeze solutii partiale sau incomplete



## Metode care lucreaza cu solutii complete

- Cautare exhaustiva
- Cautare locala
- Metoda simplex

## Metode care lucreaza cu solutii partiale/incomplete

- Algoritmi greedy
- Divide and conquer
- Programare dinamica
- Algoritmul A\*

# Cautare exhaustiva (Exhaustive search)

- Verifica fiecare solutie din spatiul de cautare pana cand solutia optima globala este gasita.
- Also called: *enumerative algorithms*
- Nu ai cum sa stii daca ai gasit solutia optima decat daca examinezi toate posibilitatile de solutie
- **Costisitor!** Ex. *TSP cu 50 de orase are  $10^{62}$  rute posibile de examinat!*
- **Totusi...**
  - Usor de implementat
  - Exista metode de a reduce volumul de munca e.g. backtracking
  - Unii algoritmi clasici de optimizare (ex. A\*, branch and bound) se bazeaza pe cautare exhaustiva

# SAT: cautare exhaustiva

- Trebuie sa generam toate sirurile posibile de n biti
- Cate sunt?
  - $2^n$
  - De la  $\langle 0 \dots 000 \rangle$  la  $\langle 1 \dots 111 \rangle$
  - Fiecare corespunde unui numar intreg
- Cum enumeram fiecare solutie posibila?
  1. Generam toate numerele intregi de la 0 la  $2^n - 1$
  2. Convertim fiecare numar intreg la sir de biti
  3. Sirul de biti este evaluat folosind o **functie de evaluare**
    - e.g. (1, daca sirul de biti satisface toate clauzele din F, 0 altfel)

**Obs:** Putem opri cautarea mai repede ex. cand o solutie satisface functia de evaluare

000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

**Am putea face o cautare mai eficienta pentru SAT?**

# TSP: cautare exhaustiva

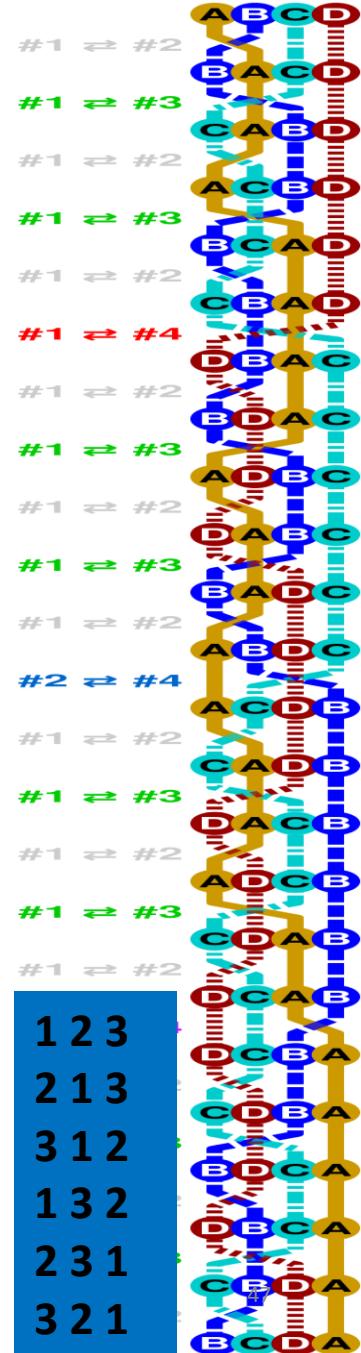
- Cum putem genera toate permutarile posibile de  $n$ ?
- Cate sunt?
  - $n!$  permutari de  $n$  numere
- Probleme
  - Daca nu fiecare oras este conectat cu toate celelalte? => Solutii nefezabile
  - Problema generarii permutarilor
- Cum enumeram fiecare solutie posibila?
  - Definirea unei functii care prioritizeaza permutarile de la 0 la  $n! - 1$
  - Recursivitate

# TSP: cautare exhaustiva

## Heap's Algorithm

```
procedure generate(n:int, P:array)
begin
    if n=1 then print P
    else begin
        for i=0; i < n-1; i=i+1 do
            generate(n-1,P)
            if n is even then
                swap(P[i],P[n-1])
            else
                swap(P[0],P[n-1])
        end for
        generate(n-1,P)
    end
end
```

- B. R. Heap, 1963
- Initial  $P[i]=i$
- Fiecare permutare este generata din precedenta prin interschimbarea a 2 elemente si nemodificarea celorlalte  $n-2$
- Metoda sistematica de a alege la fiecare pas cele 2 elemente care se interschimba

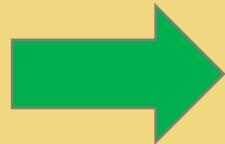


# NLP: cautare exhaustiva

- n variabile - domenii continue => o infinitate de valori posibile
- Cum putem enumera toate solutiile posibile?
  - Domeniul fiecarei variabile continue poate fi impartit intr-un numar finit de intervale
  - Se considera produsul cartezian al acestor intervale

**max.  $f(x_1, x_2)$ , unde  $x_1 \in [-1, 3]$  si  $x_2 \in [0, 12]$**

- Impartim domeniul [-1,3] in 400 intervale de lungime 0.01 fiecare
- Impartim domeniul [0,12] in 600 intervale de lungime 0.02 fiecare



$$400 \times 600 = 240,000 \text{ celule de verificat}$$

- Fiecare celula poate fi evaluata considerand un anumit punct din celula e.g. coltul celulei, mijlocul celulei

Cautarea este exhaustiva in sensul ca incercam sa acoperim toate solutiile posibile; totusi, fiecare solutie este acoperita de o celula! (cu cat mai mica cu atat mai bine)

# NLP: cautare exhaustiva

- **Dezavantaje**

- Daca folosim celule mai mici (*granularitate fina*), numarul total de celule creste semnificativ. *Ex. Daca domeniul  $x_1$  este impartit in 4000 intervale si cel al lui  $x_2$  in 6000, atunci numarul de celule creste de la 240,000 la 24,000,000*
- Cu o granularitate mica creste probabilitatea de a nu gasi cea mai buna solutie (celula care o contine nu este neaparat evaluata foarte pozitiv)
- Cand numarul de variabile este mare, aceasta cautare exhaustiva nu mai este practica pentru ca numarul de celule este mult prea mare. *Ex. O problema cu 50 variabile, fiecare cu 100 intervale, ar inseamna  $10^{100}$  celule*

- **Concluzie:** cautarea exhaustiva poate fi ok pentru probleme mici (*garanteaza gasirea celei mai bune solutii*) dar este nepractica pentru probleme mari unde nu este posibila enumerarea tuturor solutiilor posibile

# Algoritmi greedy (Greedy algorithms)

- Construiesc solutia completa intr-o serie de pasi
- La fiecare pas, se alege cea mai buna decizie disponibila (ne trebuie o euristica pentru asta)
- Alegerea de la fiecare pas este cea ***optima la nivel local***, in speranta ca vom ajunge la ***optimul global***
- La fiecare pas se alege cel mai mare “profit” – **Greedy**
- In multe probleme, o strategie greedy nu produce in general solutia optima dar poate conduce la solutii optime locale care aproximeaza solutia optima globala intr-un timp rezonabil



# SAT: Algoritmi greedy

- Luam fiecare variabila pe rand si o setam la TRUE sau la FALSE in functie de o anumita euristica
- Ce euristica ar putea ghida decizia in SAT?

Pentru fiecare variabila de la 1 la n, in orice ordine, setam valoarea de adevar care rezulta in satisfacerea celui mai mare numar de clauze care acum nu sunt satisfacute. Daca acest numar este acelasi, se alege aleator valoarea.

- ✓ Greedy
- ✓ Performanta slaba chiar si pentru probleme simple

$$\overline{x_1} \wedge (x_1 \vee x_2) \wedge (x_1 \vee x_3) \wedge (x_1 \vee x_4)$$

- Pentru  $x_1$  am alege TRUE cu 3 clauze satisfacute.
- Dar asa prima clauza nu este si nu va fi niciodata satisfacuta indiferent de valorile lui  $x_2$ ,  $x_3$  si  $x_4$  !
- **Too greedy**

# SAT: Algoritmi greedy

- Abordarea precedenta este prea greedy – ordinea in care sunt considerate variabilele nu conteaza
- Am putea incepe cu acele variabile care apar in mai putine clauze si lasa la sfarsit acele variabile des intalnite (e.g.  $x_1$ )

- Sortam variabilele in ordinea frecventei cu care apar in clauze (de la cele mai rare la cele mai frecvente)
- Pentru fiecare variabila, in ordinea de mai sus, setam valoarea de adevar care rezulta in satisfacerea celui mai mare numar de clauze care acum nu sunt satisfacute. Daca acest numar este acelasi, se alege aleator valoarea.

$$(x_1 \vee x_2) \wedge (x_1 \vee x_3) \wedge (\overline{x_1} \vee x_4) \wedge (\overline{x_2} \vee \overline{x_4}) \wedge (x_2 \vee x_5) \wedge (x_2 \vee x_6) \wedge \dots$$

Aici nu mai apar  $x_1$  si  $x_2$  dar apar restul de multe ori

- ✓  $x_1$  este prezent in 3 clauze
- ✓  $x_2$  este prezent in 4 clauze
- ✓ Toate celelalte variabile apar de mai multe ori

01.03.2022

- $x_1 = \text{TRUE}$  (2 clauze OK)
- $x_2 = \text{TRUE}$  (+ inca 2 clauze OK)
- Dar a treia si a patra clauza nu vor putea niciodata fi satisfacute in acelasi timp:  $(\overline{x_1} \vee \textcolor{red}{x_4}) \wedge (\overline{x_2} \vee \textcolor{red}{x_4})$

Camelia Chira

52

# SAT: Algoritmi greedy

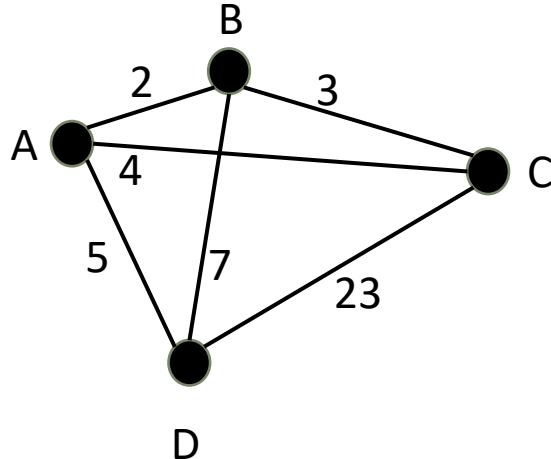
- Euristica poate fi modificata astfel incat sa adreseze problemele identificate
- *De exemplu:*
  - O regula ar putea interzice asignarea unei valori unei variabile daca oricare clauza ar deveni FALSE
  - Am putea considera cat de des apar variabile in clauzele ramase (inca nesatisfacute)
  - Sortarea variabilelor sa nu se bazeze numai pe numarul de aparitii in clauze ci si pe lungimea clauzelor in care apar (o variabila care apare intr-o clauza scurta are o alta semnificatie decat una ce apare intr-o clauza lunga cu multe alte variabile)
- Nici o alternativa nu ar insemana un algoritm care sa mearga pentru toate problemele SAT
- **Nu exista un astfel de algoritm greedy pentru SAT!**

# TSP: Algoritmi greedy

- Cel mai intuitiv algoritm greedy pentru TSP se bazeaza pe euristica **nearest-neighbor** (cel mai apropiat vecin)

Incepand de la un oras oarecare, se alege cel mai apropiat oras nevizitat inca si se continua procesul pana cand toate orasele au fost vizitate (de la ultimul oras neintoarcem in primul pentru o ruta completa).

- ✓ Greedy, dar ruta poate fi departe de una buna/perfектă
- ✓ De multe ori, pretul de platit este mare pentru alegeri greedy la inceput



- Start in A => ruta greedy: **A-B-C-D-A**
- cost (A-B-C-D-A)= $2+3+23+5=33$   
DAR
- **cost (A-C-B-D-A)= $4+3+7+5=19$**

# TSP: Algoritmi greedy

- Alte posibilitati?

Alegem urmatorul oras cel mai apropiat dar evitam situatia in care in care un oras este in mai mult de doua conexiuni (muchii pe graf).

- Incepem cu perechea de orase care are cel mai mic cost asociat.
- Continuam cu urmatoarea pereche de orase in ordinea data de cost.
- Daca am selectat (B,E) si (A,B) inseamna ca nu mai sunt permise perechi in care sa apara B.

Consideram posibilitatea de a “merge” inapoi la un oras deja vizitat => pentru fiecare pereche  $(i,j)$  de orase se alege cea mai buna optiune (e.g. sa se meargă direct de la  $i$  la  $j$  sau sa se treaca prin orasul vizitat).

*“For every common sense heuristic you can invent, you can find a pathological case that will make it look very silly.” (Michalewicz, Fogel, 2004)*

# NLP: Algoritmi greedy

- Nu există abordări greedy eficiente pentru NLP

**Un algoritm cu unele caracteristici greedy ar fi (Line search):**

- Sa presupunem ca functia de optimizat are 2 variabile  $x_1$  si  $x_2$
- Pastram  $x_1$  constanta si variem  $x_2$  pana cand ajungem la un optim
- Facem apoi noua valoare  $x_2$  gasita constanta si variem  $x_1$  pana cand un nou optim este gasit

- ✓ Abordarea nu este chiar greedy pentru ca sunt evaluate solutii complete
- ✓ Alegera celei mai bune valori pentru o dimensiune este insa o caracteristica greedy
- ✓ Probleme: performanta slabe pentru ca nu sunt considerate interactiunile dintre variabile

# Concluzii

- Metodele traditionale sunt construite pentru tipuri particulare de probleme
- Pot fi eficiente in anumite cazuri pe tipul particular de problema pentru care au fost dezvoltate
- Daca putem descompune o problema in mai multe subprobleme sau daca putem organiza spatiul de cautare ca un arbore – unele metode traditionale pot fi eficiente pe probleme de dimensiuni mici
- Dar daca asamblarea solutiilor subproblemelor este imposibila?  
Ex. TSP cu 100 orase descompus in 20 TSP cu 5 orase
- Pentru probleme reale care sunt complexe, intractabile, cu multe optime locale – avem nevoie de metode care sa mearga dincolo de ce pot face algoritmii clasici!

Cursul urmator...

Simulated Annealing

Tabu Search



**BABEŞ-BOLYAI UNIVERSITY**  
Faculty of Mathematics and Computer Science



# Inteligentă Artificială

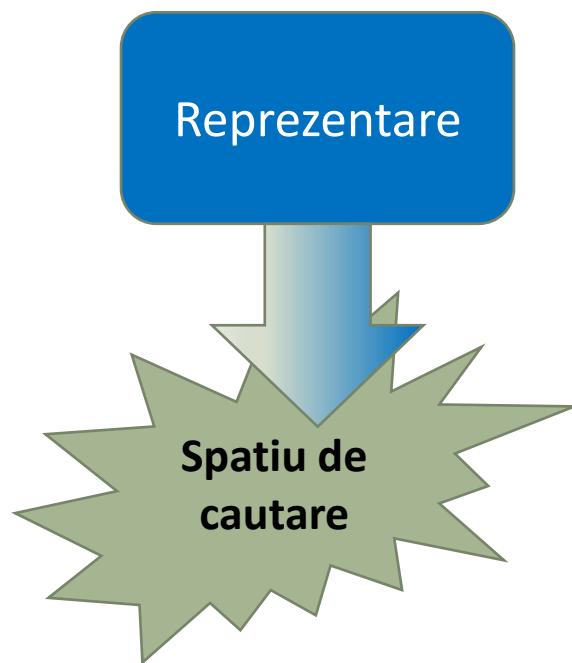
*3: Simulated Annealing, Tabu Search*

Camelia Chira

[cchira@cs.ubbcluj.ro](mailto:cchira@cs.ubbcluj.ro)

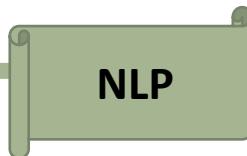
# Recap: Reprezentare

- Reprezentare comună ex. SAT, TSP, NLP
- Alte posibilități?



# Reprezentare

- Reprezentare comună ex. SAT, TSP, NLP
- Alte posibilități?



- Sir binar de lungime  $k$ ,  $k$ =numărul de biti folosiți pentru un număr real

 $\langle b_{k-1} \dots b_0 \rangle$ 

Transf. în  $x$  din  $[u, l]$

$$(\langle b_{k-1} \dots b_0 \rangle)_2 = \left( \sum_{i=0}^{k-1} b_i \cdot 2^i \right)_{10} = x'$$

- Valoarea  $x$  reală corespunzătoare cu intervalul dat  $[u, l]$

$$x = l + x' \cdot \frac{u - l}{2^k - 1}$$

**Ex:  $k=10; [u,l]=[-2,3]$**

$$\langle 1001010001 \rangle \Rightarrow x' = 593$$

$$x = -2 + 593 \cdot \frac{5}{2^{10} - 1} = 0.89833822$$

**Precizia depinde de  $k$**

Succesorul  $\langle 1001010010 \rangle \Rightarrow x = 0.90322581$  (gap almost 0.005)

# Reprezentare

- Reprezentare comună ex. SAT, TSP, NLP
- Alte posibilități?



- Permutare de  $n$  numere intregi;
- dar dacă elementul  $i$  este un număr cuprins între  $1$  și  $n-i+1$ , semnificând un oraș ramas dintr-o lista de referință  $C$ ?

Ex:  $n=9$  și  $C=(1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9)$

*Reprezentare*

(5 1 2 1 4 1 3 1 1)



5-1-3-2-8-4-9-6-7

*Ruta*

- (1) Luăm orașul nr. 5 din  $C$  (5) și apoi îl stergem din  $C$ ;  $C$  devine (1 2 3 4 6 7 8 9)
  - (2) Luăm orașul nr. 1 din  $C$  (1) și apoi îl stergem din  $C$ ;  $C$  devine (2 3 4 6 7 8 9)
  - (3) Luăm orașul nr. 2 din  $C$  (3) și apoi îl stergem din  $C$ ;  $C$  devine (2 4 6 7 8 9)
- etc.

# Reprezentare

- Reprezentare comună ex. SAT, TSP, NLP
- Alte posibilități?



- Vector de numere reale în intervalul [-1,1]

*Interpretare?*

- Valoare **TRUE** pentru numere pozitive
- Valoare **FALSE** pentru numere negative

# Metode clasice

- Metodele traditionale de rezolvare a problemelor:

- Cautare exhaustiva
- Cautare locala
- Metoda simplex

- Daca garanteaza gasirea solutiei globale, timpul de rulare este mult prea mare in rezolvarea unor probleme reale tipice
- Pot sa fie usor prinse in optime locale

- Algoritmi greedy
- Divide and conquer
- Programare dinamica
- Algoritmul A\*

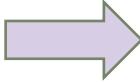
- Problemele reale tend sa fie NP-hard si sansele de a dezvolta un algoritm care sa dea rezultate in timp polinomial sunt aproape inexistente

- Avem nevoie de algoritmi capabili sa iasa din optime locale

- Simulated Annealing
- Tabu Search

# Metode de a ieși din optim local

*Introducem în algoritm:*

- Un parametru (numit **temperatura**) care schimba probabilitatea de a muta dintr-un punct în altul al spațiului de căutare  
 Simulated Annealing (SA)
- O **memorie** (istoric) care forțează un algoritm să exploreze noi zone ale spațiului de căutare  
 Tabu Search (TS)

# Local search (LS) – Simulated Annealing (SA)

```
procedure local search
begin
   $x$  = some initial starting point in  $S$ 
  while  $\text{improve}(x) \neq \text{'no'}$  do
     $x = \text{improve}(x)$ 
  return( $x$ )
end
```

Returneaza un punct  $y \in N(x)$   
daca  $y$  este mai bun decat  $x$

Cand  $y$  **nu** este mai bun decat  $x$   
retuneaza 'no' =>  $x$  este optim local

```
procedure simulated annealing
begin
   $x$  = some initial starting point in  $S$ 
  while not termination-condition do
     $x = \text{improve?}(x, T)$ 
    update( $T$ )
  return( $x$ )
end
```

Cand se termina procedura: SA depinde de  
o conditie de terminare spre deosebire de  
LS care trebuie sa gaseasca o imbunatatire

Functia **improve?**( $x, T$ ) nu returneaza  
neaparat o solutie mai buna decat  $x$ ;  
Returneaza o solutie *acceptata*

Avem un parametru  $T$  modificat periodic.  
Valoarea lui  $T$  influenteaza rezultatul  
returnat de **improve?**( $x, T$ )

# Local search (LS) – Tabu Search (TS)

```
procedure local search
begin
     $x$  = some initial starting point in  $S$ 
    while  $\text{improve}(x) \neq \text{'no'}$  do
         $x = \text{improve}(x)$ 
    return( $x$ )
end
```

Returneaza un punct  $y \in N(x)$   
daca  $y$  este mai bun decat  $x$

Cand  $y$  **nu** este mai bun decat  $x$   
retuneaza 'no' =>  $x$  este optim local

```
procedure tabu search
begin
     $x$  = some initial starting point in  $S$ 
    while not termination-condition do
         $x = \text{improve?}(x, H)$ 
        update( $H$ )
    return( $x$ )
end
```

- Structura similara cu TS
- Functia  $\text{improve?}(x, H)$  returneaza o solutie acceptata din vecinatatea lui  $x$
- Acceptarea se bazeaza pe  $H$  – istoricul cautarii
- *Solutia acceptata nu este neaparat mai buna decat  $x$*

# Iterated Hill-Climber (Iterated HC/ SAHC)

**begin**

$t = 0$

    initialize  $best$

**repeat**

$local = \text{FALSE}$

        select a current point  $c$  at random

        evaluate  $c$

**repeat**

            select all new points in the neighborhood of  $c$

            select the point  $x$  from the set of new points

                with the best value of evaluation function  $eval$

**if**  $eval(x)$  is better than  $eval(c)$  **then**  $c = x$

**else**  $local = \text{TRUE}$

**until**  $local$

$t = t + 1$

**if**  $c$  is better than  $best$  **then**  $best = c$

**until**  $t = MAX$

**end**

Numai aici iesim  
din optim local

# Stochastic HC

- Modificare Iterated HC astfel:

- In loc sa evaluam toate punctele din vecinatatea lui  $c$  si sa il selectam apoi pe cel mai bun -> selectam un singur punct  $x$  din vecinatate
- Acceptam acest punct  $x$  cu o probabilitate care depinde de calitatea punctelor  $c$  si  $x$  (diferenta dintre functia de evaluare a lui  $c$  si  $x$ )

=> Stochastic Hill-Climber

**begin**

$t = 0$

select a current point  $c$  at random

evaluate  $c$

**repeat**

select the point  $x$  from the neighborhood of  $c$

select  $x$  with probability  $\frac{1}{1+e^{\frac{eval(c)-eval(x)}{T}}}$

$t = t+1$

**until**  $t = MAX$

Un singur  
repeat: nu  
trebuie sa  
repetam  
iteratia  
incepand din  
alt punct

Un punct nou selectat  
este acceptat cu o  
anumita probabilitate

# Stochastic HC

$$p = \frac{1}{1 + e^{\frac{eval(c) - eval(x)}{T}}}$$

$p$  = probabilitatea de a accepta un punct nou depinde de:

1. Diferenta dintre  $eval(c)$  si  $eval(x)$
2. Un parametru  $T$  (care este constant in timpul executiei algoritmului)

Care este rolul parametrului  $T$ ?

Sa presupunem ca avem o problema de maximizare:

$$eval(c) = 107$$

$$eval(x) = 120$$

$$eval(c) - eval(x) = -13 \Rightarrow x \text{ este mai bun decat } c$$

*Cu ce probabilitate este acceptat  $x$  pe baza lui  $T$ ?*

$T$	$e^{\frac{-13}{T}}$	$p$
1	0.000002	1.00
5	0.0743	0.93
10	0.2725	0.78
20	0.52	0.66
50	0.77	0.56
$10^{10}$	0.9999...	0.5...

$T$	$e^{\frac{-13}{T}}$	$p$
1	0.000002	1.00
5	0.0743	0.93
10	0.2725	0.78
20	0.52	0.66
50	0.77	0.56
$10^{10}$	0.9999...	0.5...

# Stochastic HC

- Cu cat  $T$  este mai mic cu atat este mai putin importanta evaluarea punctelor
- $T$  este foarte mare =>  $p$  se apropie de 0.5 => random search!
- $T$  este foarte mic => iterated HC

Alegerea valorii lui  $T$  este importantă

$T=10$  ,  $\text{eval}(c) = 107$

Daca  $x$  este mai slab  $p$  scade

Daca  $x$  are aceeasi calitate cu  $c$ , atunci probabilitatea de acceptare este 0.5 – ok!

Daca  $x$  este mai bun  $p$  creste

Probabilitatea de acceptare  $p$  in functie de  $x$

$\text{eval}(x)$	$\text{eval}(c)-\text{eval}(x)$	$e^{\frac{\text{eval}(c)-\text{eval}(x)}{T}}$	$p$
80	27	14.88	0.06
100	7	2.01	0.33
107	0	1.00	0.50
120	-13	0.27	0.78
150	-43	0.01	0.99

# Simulated Annealing (SA)

- Algoritmul SA este similar cu Stochastic HC
- Principala diferență: **parametrul T se schimba in timpul rularii**
- SA incepe cu valori mari ale lui T (*random search la inceput*) și scade gradual valoarea lui T (*iterated HC la sfarsit*)
- În plus, SA acceptă întotdeauna punctele mai bune decât cel curent

## SA – analogie din termodinamica

Physical System	Optimization Problem
state	feasible solution
energy	evaluation function
ground state	optimal solution
rapid quenching	local search
temperature	control parameter $T$
careful annealing	simulated annealing



a.k.a

Monte Carlo annealing  
Statistical cooling  
Probabilistic hill-climbing  
Stochastic relaxation  
Probabilistic exchange algorithm

# SA Procedure

**begin**

$t = 0$

    initialize T

    select a current point  $c$  at random

    evaluate  $c$

**repeat**

**repeat**

            select a new point  $x$  from the neighborhood of  $c$

**if**  $\text{eval}(c) < \text{eval}(x)$

**then**  $c \leftarrow x$

**else if**  $\text{random}[0,1] < e^{\frac{\text{eval}(x)-\text{eval}(c)}{T}}$  **then**  $c \leftarrow x$

**until** (termination-condition)

$T \leftarrow g(T, t)$

$t \leftarrow t + 1$

**until** (halting-criterion)

**end**

# SA checklist

- Intrebari specifice problemei:
  - Ce este o solutie?
  - Care sunt vecinii unei solutii?
  - Care este costul unei solutii?
  - Cum se determina solutia initiala?

⇒ Structura spatiului de cautare si a vecinatatii, functia de evaluare

- SA aduce intrebari aditionale:
  - *Cum determinam valoarea initiala a temperaturii  $T$ ?*
  - *Cum determinam rata de racire  $g(T,t)$ ?*
  - *Cum setam conditia de terminare a iteratiei interioare (termination-condition)?*
  - *Cum setam conditia de oprire (halting-criterion)?*

# SA steps

PAS 1:

$$T \leftarrow T_{max}$$

select  $c$  aleator

PAS 2:

select  $x$  din vecinatatea lui  $c$

**if** eval( $x$ ) e mai bun decat eval( $c$ )

**then** select  $x$  ( $c \leftarrow x$ )

**else** select  $x$  cu probabilitatea  $e^{\frac{eval(x)-eval(c)}{T}}$

**repeat** step 2 de  $k_T$  ori

PAS 3:

set  $T \leftarrow rT$

**if**  $T \geq T_{min}$  **then** goto PAS 2

**else** goto PAS 1

## Parametri de setat:

- $T_{max}$  (temperatura initiala)
- $k_T$  (numarul de iteratii)
- $r$  (rata de racire)
- $T_{min}$  (temperatura de inghet)

# SA-SAT

La inceputul acestui **repeat**  
 $T = T_{max}$  (cum  $j=0$ ) si  $v$  = aleator

```
procedure SA-SAT
begin
    tries ← 0
    repeat
        v ← random truth assignment
        j ← 0
        repeat
            if v satisfies the clauses then return v
             $T = T_{max} \cdot e^{-j \cdot r}$ 
            for k = 1 to the number of variables do
                begin
                    compute the increase (decrease)  $\delta$  in the
                    number of clauses made true if  $v_k$  was flipped
                    flip variable  $v_k$  with probability  $(1 + e^{-\frac{\delta}{T}})^{-1}$ 
                    v ← new assignment if the flip is made
                end
            j ← j + 1
        until  $T < T_{min}$ 
        tries ← tries + 1
    until tries = MAX_T
end
```

## LS vs. SA-SAT

- LS poate sa faca un pas inapoi (i.e descreste nr. de clauze FALSE) daca alte mutari nu sunt posibile
- SA-SAT poate face un numar arbitrar de 'pasii inapoi'

**SA-SAT poate sa iasa din optime locale!**

- Se incercă diferite valori TRUE/FALSE în  $v$  pentru fiecare variabilă (*flip cu prob. p*)
- $p$  depinde de  $\delta$  (improvement of flip) și  $T$

# Parametrii SA-SAT

**r = rata de scadere a temperaturii**

$T$  scade de la  $T_{max}$  la  $T_{min}$  prin  
incrementarea lui  $j$ :  $T = T_{max} \cdot e^{-j \cdot r}$

**Valori folosite (Spears, 1996):**

$$T_{max} = 0.30$$

$$T_{min} = 0.01$$

$$r = \frac{1}{N \cdot tries}$$

```
procedure SA-SAT
begin
    tries ← 0
repeat
    v ← random truth assignment
    j ← 0
repeat
    if v satisfies the clauses then return v
     $T = T_{max} \cdot e^{-j \cdot r}$ 
    for  $k = 1$  to the number of variables do
begin
    compute the increase (decrease)  $\delta$  in the
        number of clauses made true if  $v_k$  was flipped
    flip variable  $v_k$  with probability  $(1 + e^{-\frac{\delta}{T}})^{-1}$ 
    v ← new assignment if the flip is made
end
    j ← j + 1
until  $T < T_{min}$ 
tries ← tries + 1
until tries = MAX-TRIES
end
```

# TSP: Simulated Annealing

- SA in care {
  - $c = \text{ruta}$
  - $\text{eval}(c) = \text{lungimea drumului pe ruta } c$

```
begin
     $t = 0$ 
    initialize T
    select a current point  $c$  at random
    evaluate  $c$ 
repeat
    repeat
        select a new point  $x$  from the neighborhood of  $c$ 
        if  $\text{eval}(c) < \text{eval}(x)$ 
            then  $c \leftarrow x$ 
        else if  $\text{random}[0,1] < e^{\frac{\text{eval}(x)-\text{eval}(c)}{T}}$  then  $c \leftarrow x$ 
    until (termination-condition)
     $T \leftarrow g(T, t)$ 
     $t \leftarrow t + 1$ 
until (halting-criterion)
end
```

# TSP: Simulated Annealing

Diferentele dintre diferitele abordari SA pentru TSP vin din:

- Metoda de a genera solutia initiala
- Definitia vecinatatii unei rute
- Selectarea unui vecin
- Metoda de a descreste temperatura
- Conditia de terminare inner-loop (termination)
- Conditia de oprire a algoritmului (halting-criterion)
- Existenta unei faze postprocesare

# TSP: Simulated Annealing

## EXEMPLU PSEUDOCOD

```
T = 10000; alpha = 0.9999; minT = 0.00001;  
c = createRandomSolution();  
while (T > minT)  
    repeat  
        x = GetVecin(c); // swap 2 cities / 2-opt / etc  
        delta = eval(x) - eval(c);  
        if (delta < 0) then c = x  
        else if random.NextDouble() < Math.Exp(-delta/T) then c=x  
    until (max-iterations)  
    T = alpha*T;  
end while  
return c
```

# NLP: Simulated Annealing

- SA este usor de aplicat in optimizare numérica
- Variabile continue => vecinatatea poate fi definită pe baza unei distributii Gauss (pentru fiecare variabilă)
- Punct curent  $x$

$$x = (x_1, x_2, \dots, x_n) \in \mathbb{R}^n$$

$$l_i \leq x_i \leq u_i, 1 \leq i \leq n$$

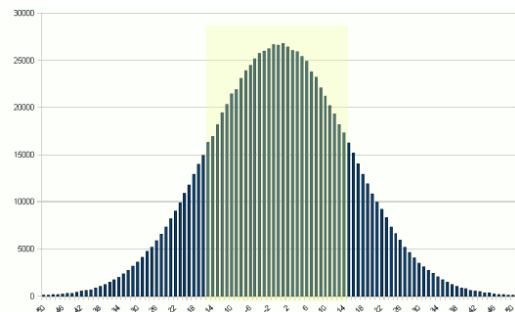
- Vecinul  $x'$  a lui  $x$  este:

$$x'_i \leftarrow x_i + N(0, \sigma_i)$$

unde  $\sigma_i = (u_i - l_i)/6$

$N(0, \sigma_i)$  este aleator ales din distribuția

Gauss (mean 0, deviație standard  $\sigma_i$ )



# NLP: Simulated Annealing

- Schimbarea in functia de evaluare poate fi calculata:

$$\Delta eval = eval(x) - eval(x')$$

- Daca  $\Delta eval > 0$  (problema de minimizare):

- Noul punct  $x'$  se accepta ca noua solutie
- Altfel,  $x'$  este acceptat cu probabilitatea  $e^{\frac{\Delta eval}{T}}$

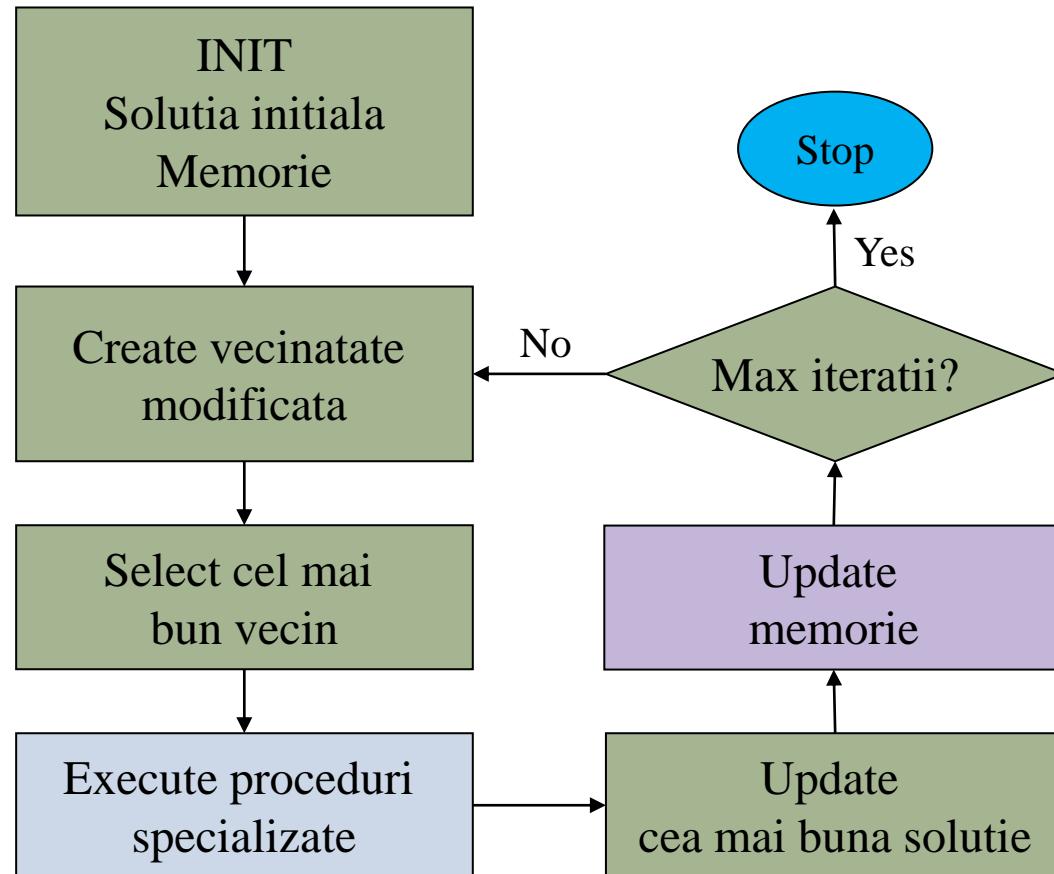
**Decizii SA pentru orice problema de optimizare numerica:**

- Solutia initiala
  - Vecinatate
  - Selectarea unui vecin
- > **usor de luat (i.e. random, Gaussian)**
- Metoda de a modifica temperatura
  - Conditia de terminare inner-loop
  - Conditia de oprire a algoritmului
  - Existenta unei faze postprocesare

# Tabu Search

- Idee principală: să pastrăm o “memorie” care să forteze căutarea să exploreze noi zone ale spațiului de căutare
- Putem retine unele soluții examineate deja => ele devin puncte **tabu** (interzise) din spațiul de căutare
- Spre deosebire de SA, este o metodă determinista (fiind însă posibil să ii adaugăm elemente probabiliste)

# Tabu Search



# Tabu Search

Given a feasible solution  $x^*$  with objective function value  $f^*$ :

Let  $x = x^*$  with  $f(x) = f^*$ .

**while** stopping criterion is not fulfilled **do**

**begin**

(1) select best admissible move that transforms  $x$  into  $x'$  with objective function value  $f(x')$  and add its attributes to the running list

(2) perform tabu list management: compute moves (or attributes) to be set tabu, i.e., update the tabu list

(3) perform exchanges:  $x = x'$ ,  $f(x) = f(x')$ ;

if  $f(x) < f^*$  then  $f^* = f(x)$ ,  $x^* = x$

**end**

Result:  $x^*$  is the best of all determined solutions, with objective function value  $z^*$ .

# SAT: Tabu Search

- SAT cu 8 variabile  $x_1, x_2, \dots, x_8$
- Pentru formula logica F, cautam valorile TRUE sau FALSE pentru fiecare variabila  $x_1, x_2, \dots, x_8$  astfel incat F = TRUE

O solutie initiala este  $x = (x_1, x_2, \dots, x_8)$   
i.e.  $x=(0,1,1,1,0,0,0,1)$

Functia de evaluare: suma ponderata a numarului de clauze TRUE (ponderile sunt date de numarul de variabile din clauza)  
Ex.  $\text{eval}(x) = 27$

Vecinatate: 8 solutii - fiecare obtinuta prin schimbarea unui bit (flip) din x  
Obs: HC search inseamna evaluarea celor 8 solutii vecine si selectarea celei mai bune

$$x=(0,1,1,1,0,0,0,1)$$

N(x):

$$\begin{aligned}x_{n_1} &= (1,1,1,1,0,0,0,1) \\x_{n_2} &= (0,0,1,1,0,0,0,1) \\x_{n_3} &= (0,1,0,1,0,0,0,1) \\x_{n_4} &= (0,1,1,0,0,0,0,1) \\x_{n_5} &= (0,1,1,1,1,0,0,1) \\x_{n_6} &= (0,1,1,1,0,1,0,1) \\x_{n_7} &= (0,1,1,1,0,0,1,1) \\x_{n_8} &= (0,1,1,1,0,0,0,0)\end{aligned}$$

# SAT: Tabu Search

$x=(0,1,1,1,0,0,0,1)$

$eval(x) = 27$

Sa presupunem ca flip bit 3  $\Rightarrow$  cea mai buna evaluare i.e. 31

- Tabu Search: memorie
- Retinem indexul variabilei schimbate si iteratia cand s-a intamplat

Memoria este un vector M initializat la 0 si modificat la pozitia i astfel:

$$M(i) = j, (\text{and } j \neq 0)$$

Interpretare:  $j$  este cea mai recenta iteratie in care bitul  $i$  a fost schimbat  
 $j=0$  inseamna ca bitul  $i$  nu a fost niciodata schimbat

- *Folositor*: dupa o perioada de timp (nr iteratii), informatia din memorie se sterge
- Interpretarea se poate schimba astfel: *pe pozitia i din M pastram numarul de iteratii ramas in care schimbarea bitului i nu este permisa (tabu list)*

# SAT: Tabu Search

$x=(0,1,1,1,0,0,0,1)$

$eval(x) = 27$

Sa presupunem ca flip bit 3  $\Rightarrow$  cea mai buna evaluare i.e. 31

Sa presupunem ca orice informatie poate sta in memorie 5 iteratii.

Memoria:

$$M(i) = j, (\text{and } j \neq 0)$$

Interpretare: bitul  $i$  a fost schimbat acum 5-j iteratii

In exemplul nostru, dupa schimbarea bitului 3:

- $x=(0,1,0,1,0,0,0,1)$
- $eval(x) = 31$
- $M = \boxed{0 \ 0 \ 5 \ 0 \ 0 \ 0 \ 0 \ 0}$

$M(3)=5 \Leftrightarrow$  pentru urmatoarele 5 iteratii pozitia 3 nu este disponibila (tabu)

# SAT: Tabu Search

Dupa alte 4 iteratii in care selectam cel mai bun vecin disponibil (nu neaparat si cel mai bun din toti – dar asa iesim de fapt din optime locale!) avem:

$$M = \boxed{3 \ 0 \ 1 \ 5 \ 0 \ 4 \ 2 \ 0} \quad M \text{ dupa 5 iteratii}$$

- Bitii 2, 5 si 8 sunt disponibili si pot fi schimbat oricand
  - Bitul 1 nu este disponibil inca 3 iteratii
  - Bitul 3 nu este disponibil inca 1 iteratie
  - Bitul 4 a fost schimbat chiar in aceasta iteratie si nu va fi disponibil urmatoarele 5 iteratii, etc.
- ✓ Cea mai recenta schimbare:  $M(4)=5$  (bitul 4 schimbat acum  $5-5=0$  iteratii)
- ✓ Celelalte schimbari in ordine inversa:  $M(6)=4$ ,  $M(1)=3$ ,  $M(7)=2$ ,  $M(3)=1$

⇒ **Solutia curenta este  $x=(1,1,0,0,0,1,1,1)$**

Prespunem ca  $eval(x)=33$

# SAT: Tabu Search

$x=(1,1,0,0,0,1,1,1)$

$eval(x)=33$

$M= \begin{array}{|c|c|c|c|c|c|c|c|} \hline 3 & 0 & 1 & 5 & 0 & 4 & 2 & 0 \\ \hline \end{array}$

N(x):

$xn_1=(0,1,0,0,0,1,1,1)$

$xn_2=(1,0,0,0,0,1,1,1)$

$xn_3=(1,1,1,0,0,1,1,1)$

$xn_4=(1,1,0,1,0,1,1,1)$

$xn_5=(1,1,0,0,1,1,1,1)$

$xn_6=(1,1,0,0,0,0,1,1)$

$xn_7=(1,1,0,0,0,1,0,1)$

$xn_8=(1,1,0,0,0,1,1,0)$

- TS evalueaza fiecare vecin dar forteaza cautarea sa exploreze alte zone din spatiu
- **Cum?** – Schimbarile retinute in M sunt cele mai recente flips si sunt interzise in selectarea noii solutii

**Iteratia 6:**

- Nu este permis sa schimbam bitii 1, 3, 4, 6 si 7
  - Solutiile vecine obtinute din flips interzise sunt tabu
- ⇒ ***Solutia urmatoare este selectata din schimbari poz. 2, 5, 8***

- Sa presupunem ca cea mai buna evaluare este data de schimbare la poz 5
- $eval(xn_5)=32$

$\Rightarrow M= \begin{array}{|c|c|c|c|c|c|c|c|} \hline 2 & 0 & 0 & 4 & 5 & 3 & 1 & 0 \\ \hline \end{array}$

# SAT: Tabu Search

- TS continua cu iteratii similare pana la un numar maxim de iteratii
- In orice etapa, exista o solutie curenta care se examineaza impreuna cu vecinatatea ei din care solutiile tabu sunt eliminate
- **Dar daca una din solutiile tabu are o evaluare excelenta? Poate cautarea ar trebui sa fie mai flexibila!**

## Aspiration Criterion

- Se evaluateaza toti vecinii si in circumstante normale, se selecteaza cea mai buna solutie non-tabu
- Dar atunci cand o solutie foarte bine evaluata este gasita intre vecini (circumstante care nu sunt “normale”) atunci aceasta solutie este selectata

# Tabu Search: flexibilitate

**Alte modalitati de face cautarea mai flexibila:**

- Selectia determinista sa fie modificata intr-un probabilista (in care solutii mai bune sa aiba sanse mai mari sa fie selectate)
- Schimbarea orizontului memoriei: uneori este util sa fie mai mare, alteori e mai bine sa fie mai mic (e.g. cand este in hill-climbing area)
- **Introducerea unei memorii de lunga durata!**

# TS cu long-term memory

- M inregistreaza actiuni din ultimele cateva iteratii -> recency-based memory (sau short-term memory)
- Daca am extinde M la un frequency-based memory (long-term memory) ?

Memoria long-term este un vector H initializat la 0 si modificat la pozitia i astfel:

$$H(i) = j$$

Interpretare: in timpul ultimelor h iteratii bitul i a fost modificat de j ori

- Valoarea lui h este mare
- Ex: Dupa 100 de iteratii cu h=50, memoria de lunga durata este:

5	7	11	3	9	8	1	6
01.03.2022							

Camelia Chira

- H poate fi folosita pentru a diversifica cautarea
- **Cum?**
  - H ne spune care biti au fost schimbuti de foarte putine ori
  - Diversificare => explorarea acelor posibilitati

# TS cu long-term memory

- Memoria de lunga durata se foloseste de obicei in cazuri speciale
  - Daca toate solutiile non-tabu conduc spre o solutie mai slaba
- Exista multe feluri in care informatiile oferite de memoria de lunga durata sunt folosite in luarea deciziilor
- De obicei:
  - Cea mai frecventa schimbare din memoria de lunga durata devine cea mai putin atractiva
  - Valoarea data de functia de evaluare este diminuata cu o masura de penalizare care depinde de frecventa observata in H

# SAT: TS cu long-term memory

- Cea mai buna solutie gasita pana acum are eval. 37
- Solutia curenta este  $x$ :  $eval(x)=35$
- Non-tabu flips sunt posibili la poz. 2, 3, 7
- Evaluariile coresp. sunt 30, 33, 31
- Aspiration criterion nu poate fi aplicat => folosim H
- Pentru o noua solutie  $x'$  evaluarea este:

$$eval(x') - penalty(x')$$

unde  $penalty(x')=0.7*H(i)$ , 0.7 este un coefficient

H =	5	7	11	3	9	8	1	6
-----	---	---	----	---	---	---	---	---

Flip bit	eval( $x'$ )	$eval(x') - penalty(x')$
2	30	$30 - 0.7 * 7 = 25.1$
3	33	$33 - 0.7 * 11 = 25.3$
7	31	$31 - 0.7 * 1 = 30.3$

# Tabu Search

- Exista multe alte optiuni ce pot fi folosite in TS
- **Aspiration by default:**
  - Cand trebuie aleasa o solutie tabu, sa fie selectata dintre cele mai “vechi” considerate
- **Aspiration by search direction:**
  - Se memoreaza nu numai un set de miscari recente ci si daca acestea au insemnat sau nu o imbunatatire a solutiei
- **Aspiration by influence:**
  - Influence masoara gradul de schimbare a unei solutii noi (e.g. distanta dintre solutia noua si cea veche)
  - O miscare noua are o influenta mai mare daca s-a facut un pas mai mare de la vechea solutie la noua solutie

# TSP: Tabu Search

- 8-city TSP

O solutie initiala este o permutare de la 1 la 8  
 $x=(2,4,7,8,1,5,3,6)$

Functia de evaluare: distanta totala intre orase in ordinea data de permutare

Vecinatate: un vecin obtinut prin interschimbarea a 2 orase (swap)  
28 solutii in vecinatate:  $\exists \binom{8}{2} = \frac{7 \cdot 8}{2 \cdot 1} = 28$  perechi de orase ce pot fi interschimbat

Care poate fi atunci structura memoriei?

# TSP: Tabu Search

- M trebuie sa retina de cate ori in ultimele iteratii un anumit swap intre doua orase i si j a avut loc

- swap(i,j)*** este retinuta in linia i, coloana j ( $i < j$ )
- i si j sunt orase, nu pozitii in permutare

- In M retinem istoricul ultimelor 5 iteratii
- H poate avea aceeasi structura ca si M
- H retine istoricul ultimelor h=50 iteratii

2	3	4	5	6	7	8
1						
2						
3						
4						
5						
6						
7						

# TSP: Tabu Search

- M si H au fost initializate la 0
- Dupa **500 iteratii**, solutia curenta este  $x=(7,3,5,6,1,2,4,8)$  si  $\text{eval}(x)=173$
- Cea mai buna solutie de pana acum are eval. 171

**M**

2	3	4	5	6	7	8
0	0	1	0	0	0	0
0	0	0	5	0	0	0
0	0	0	4	0	0	0
3	0	0	0	0	0	0
0	0	2	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

**H**

1	2	3	4	5	6	7	8	1
0	2	3	3	0	1	1	0	1
2	1	3	1	1	1	0	0	2
2	3	3	3	4	0	0	0	3
1	1	2	1	1	2	1	1	4
4	2	1	4	2	1	0	0	5
3	1	3	3	2	1	0	0	6
6	3	1	0	6	1	0	0	7
7	6	0	0	0	0	0	0	0

# TSP: Tabu Search

- M si H au fost initializate la 0
- Dupa **500 iteratii**, solutia curenta este  $x=(7,3,5,6,1,2,4,8)$  si  $\text{eval}(x)=173$
- Cea mai buna solutie de pana acum are eval. 171

M

2	3	4	5	6	7	8
0	0	1	0	0	0	0
0	0	0	5	0	0	0
0	0	0	4	0	0	0
3	0	0	0	0	0	0
0	0	2	0	0	0	0
0	0	0	0	0	0	0

1  
2  
3  
4  
5  
6  
7

M(2,6)=5 indica cel mai recent swap  
 $\Rightarrow$  solutia precedenta era  $(7,3,5,2,1,6,4,8)$   
 $\Rightarrow \text{swap}(2,6)$  este tabu pentru 5 iteratii

Tabu list:

- $\text{swap}(1,4)$
- $\text{swap}(2,6)$
- $\text{swap}(3,7)$
- $\text{swap}(4,5)$
- $\text{swap}(5,8)$

Cea mai veche

Numai 5 swaps din 28 posibile sunt interzise (tabu)

# TSP: Tabu Search

- M si H au fost initializate la 0
- Dupa **500 iteratii**, solutia curenta este  $x=(7,3,5,6,1,2,4,8)$  si  $\text{eval}(x)=173$
- Cea mai buna solutie de pana acum are eval. 171

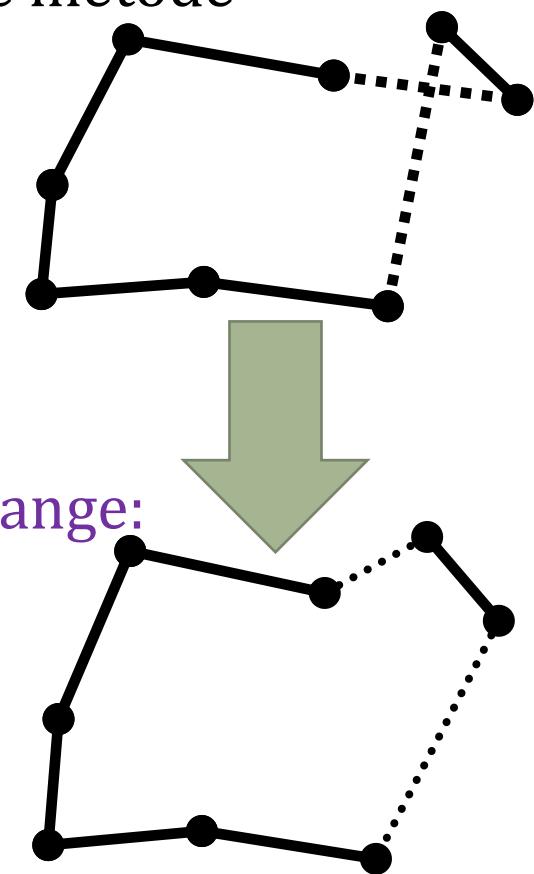
H							
2	3	4	5	6	7	8	1
0	2	3	3	0	1	1	2
2		1	3	1	1	0	3
2	3		3	4	0		4
1	1	2					5
4	2	1					6
3	1						7
6							

- $\text{swap}(7,8)$  este cel mai frecvent: orasele 7 si 8 au fost interschimbate de 6 ori in ultimele 50 iteratii
- Exista perechi de orase care nu au fost deloc interschimbate in ultimele 50 iteratii e.g. (1,6), (2,8),(3,8)

# TSP: Tabu Search

Rutele vecine pot fi determinate folosind alte metode

- 2-interchange move (2-opt)
  - multimea rutelor obtinute prin interschimbarea a doua muchii neadiacente
  - Stergem 2 muchii neadiacente
  - Adaugam alte 2 muchii astfel incat sa obtinem o ruta valida
- Abordare TS folosind vecinatatea 2-interchange:
  - O ruta devine tabu daca ambele muchii din interschimbare se aflau in lista tabu
  - Lista tabu este updatata adaugand muchiile adaugate in lista (si ignorand muchiile sterse)
  - Lista tabu are lungime fixa



# TSP: TS algorithm with 2-interchange moves

```
procedure tabu search
```

```
begin
```

```
    tries ← 0
```

```
repeat
```

```
    generate a tour
```

```
    count ← 0
```

```
repeat
```

```
    identify a set  $\mathcal{T}$  of 2-interchange moves
```

```
    select the best admissible move from  $\mathcal{T}$ 
```

```
    make appropriate 2-interchange
```

```
    update tabu list and other variables
```

```
    if the new tour is the best-so-far for a given ‘tries’
```

```
        then update local best tour information
```

```
        count ← count +1
```

```
until count = ITER
```

```
tries ← tries +1
```

```
if the current tour is the best-so-far (for all ‘tries’)
```

```
    then update global best tour information
```

```
until tries = MAX-TRIES
```

```
end
```

## Parametri

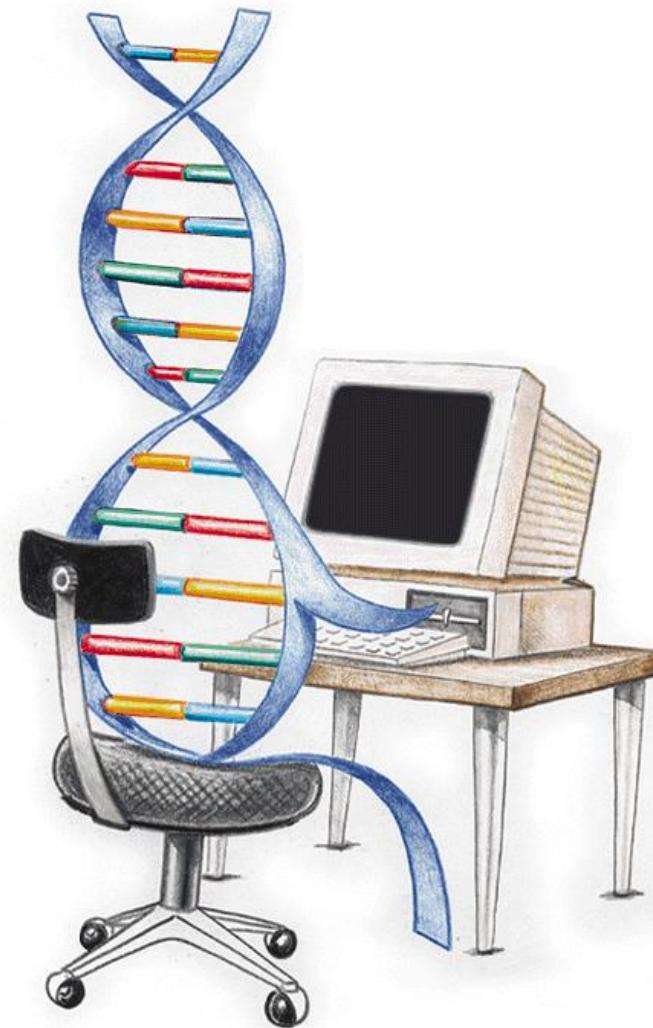
- Lungime tabu list =  $3n$
- Pentru  $n=100$ :
  - MAX-TRIES=4
  - Iter= $0.0003 \cdot n^4$

# Concluzii

Simulated Annealing	Tabu Search
<i>Both designed to escape local optima</i>	
✓ Can make uphill moves at any time	✓ Makes uphill moves only when it is stuck in local optima
✓ Stochastic algorithm	✓ Deterministic algorithm
<p>✓ Lucreaza cu solutii complete (ca si algoritmii clasici de cautare exhaustiva si cautare locala)</p> <p>✓ Pot fi opriti in orice iteratie</p>	
<ul style="list-style-type: none"><li>■ Au multi parametri de setat e.g. temperatura, rata de racire, memorie, etc.</li></ul>	

Next time...

# Algoritmi Evolutivi





**BABEŞ-BOLYAI UNIVERSITY**  
Faculty of Mathematics and Computer Science



# Inteligentă Artificială

*4: Algoritmi Evolutivi*

**Camelia Chira**

[cchira@cs.ubbcluj.ro](mailto:cchira@cs.ubbcluj.ro)



# Am vazut deja ca...

- Metodele traditionale de rezolvare a problemelor
  - Daca garanteaza gasirea solutiei globale, timpul de rulare este mult prea mare in rezolvarea unor probleme reale tipice
  - Pot sa fie usor prinse in optime locale
- Problemele reale tind sa fie NP-hard si sansele de a dezvolta un algoritm care sa dea rezultate in timp polinomial sunt aproape inexistente
  - Cautare exhaustiva
  - Cautare locala
  - Metoda simplex
  - Algoritmi greedy
  - Divide and conquer
  - Programare dinamica
  - Algoritmul A\*
- Algoritmi capabili sa iasa din optime locale
  - Simulated Annealing
  - Tabu Search

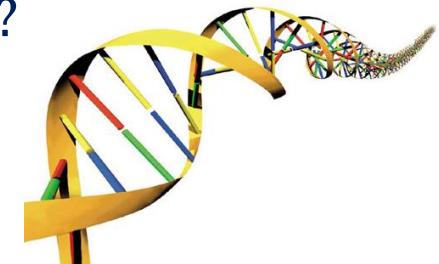


# Recap

- Toate aceste metode se bazeaza pe o singura solutie care este explorata in fiecare iteratie
- Ideea este simpla si intuitiva, poate da rezultate bune de multe ori
- Putem folosi:
  - **Reguli deterministe** ex. *Hill-climbing: daca un vecin este mai bun, il selectam si continuam cautarea de acolo; altfel cautam in vecinatatea punctului curent*
  - **Reguli probabiliste** ex. *Simulated Annealing: daca un vecin este mai bun, il selectam; altfel, acceptam cu o anumita probabilitate aceasta pozitie mai slaba sau continuam cautarea in vecinatatea curenta*
  - **Istoricul cautarii** ex. *Tabu Search: selectam cel mai bun vecin care nu trebuie neaparat sa fie mai bun decat solutia curenta dar trebuie sa nu fie in lista tabu*
- Dar daca am mentine mai multe solutii simultan?

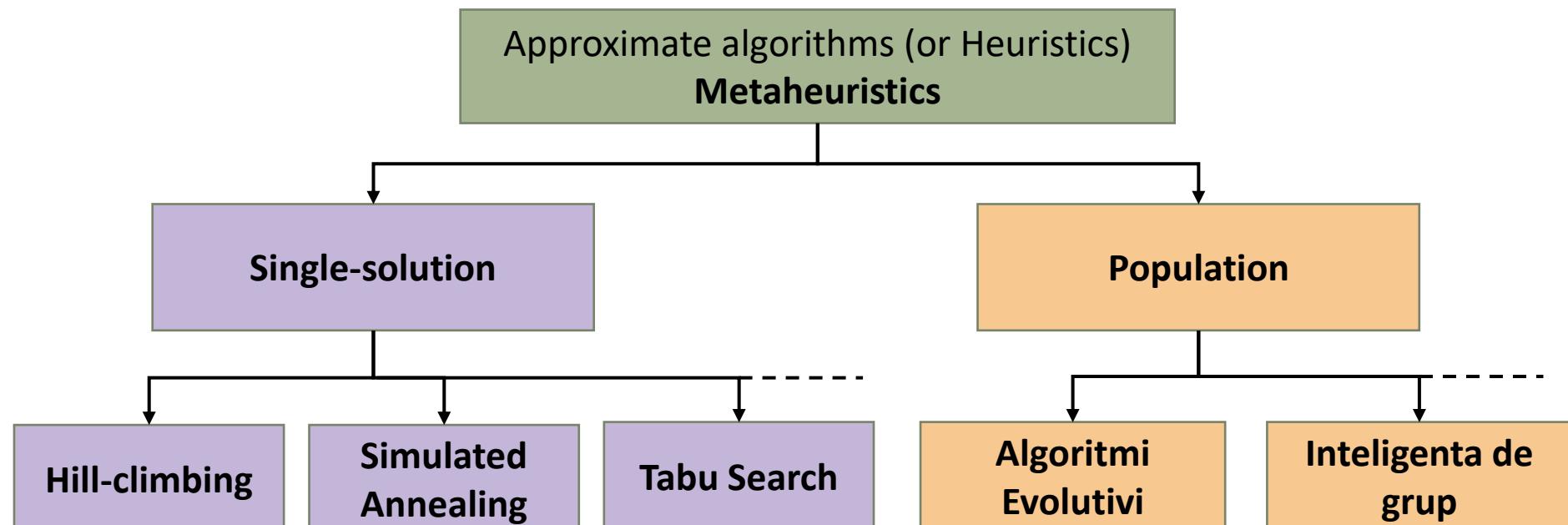
Daca am avea o populatie de solutii?

## Algoritmi Evolutivi



# Metode de optimizare

- Algoritmi exacti e.g. Branch and bound, Dynamic Programming
  - **Obtin solutia optima globala**
  - Probleme de dimensiuni mici
- Algoritmi aproximativi
  - **Nu garanteaza solutia optima globala**



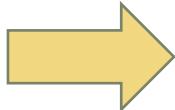
# Dezvoltarea metaeuristicilor

- Nu există o super metodă pentru toate problemele de optimizare  
**No-Free Lunch Theorem**
- Echilibre:
  - *Diversificare – Intensificare*
  - *Exploatare – Explorare*

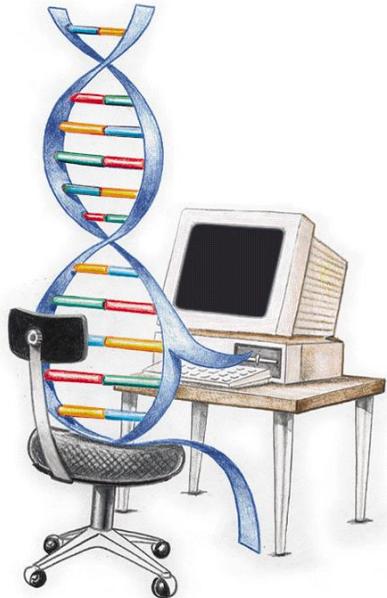


# Algoritmi Evolutivi (AE)

- Algoritmi de cautare care lucreaza cu *populatii* de solutii

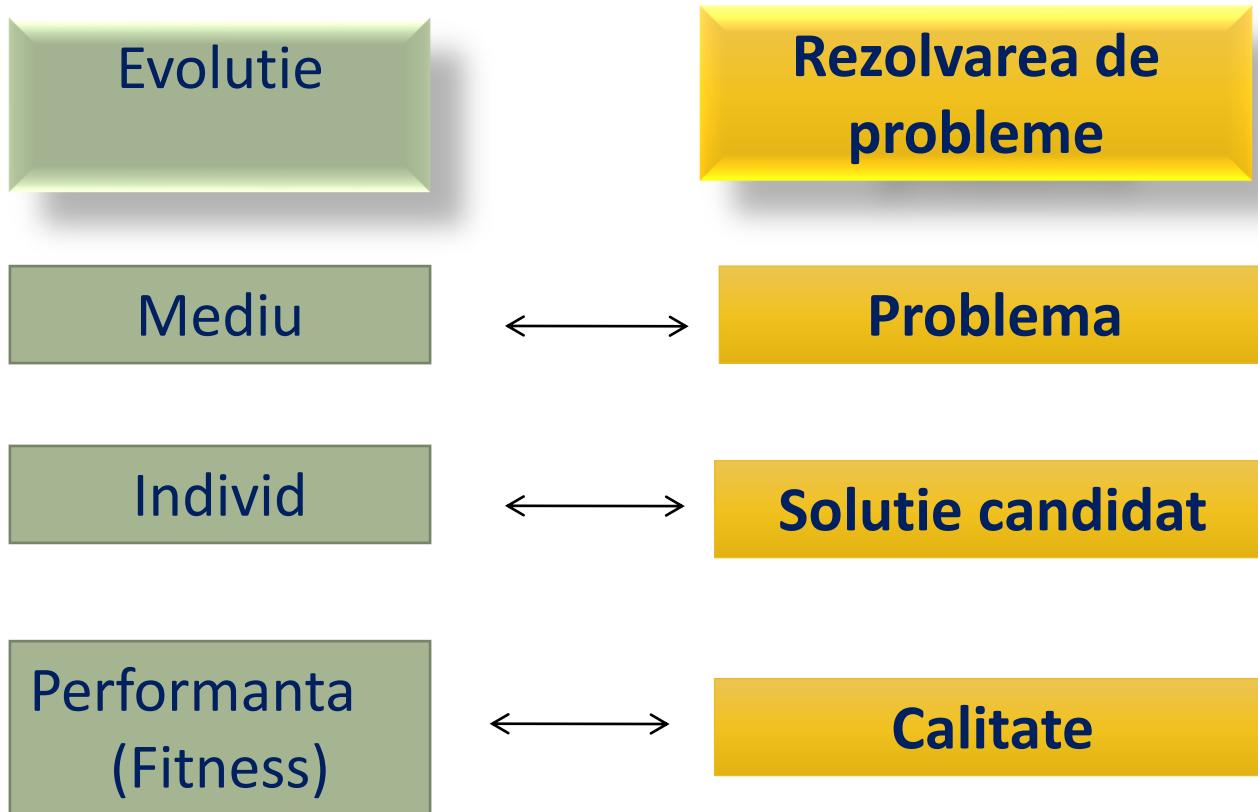


*Un concept nou:*  
**Competitie** intre solutiile din populatie



- Simularea unor procese evolutive de competitie si selectie
- *Solutiile candidate lupta pentru un loc in generatiile viitoare!*

# Calcul Evolutiv (Evolutionary Computing)



**Fitness** – sansa de supravietuire si reproducere

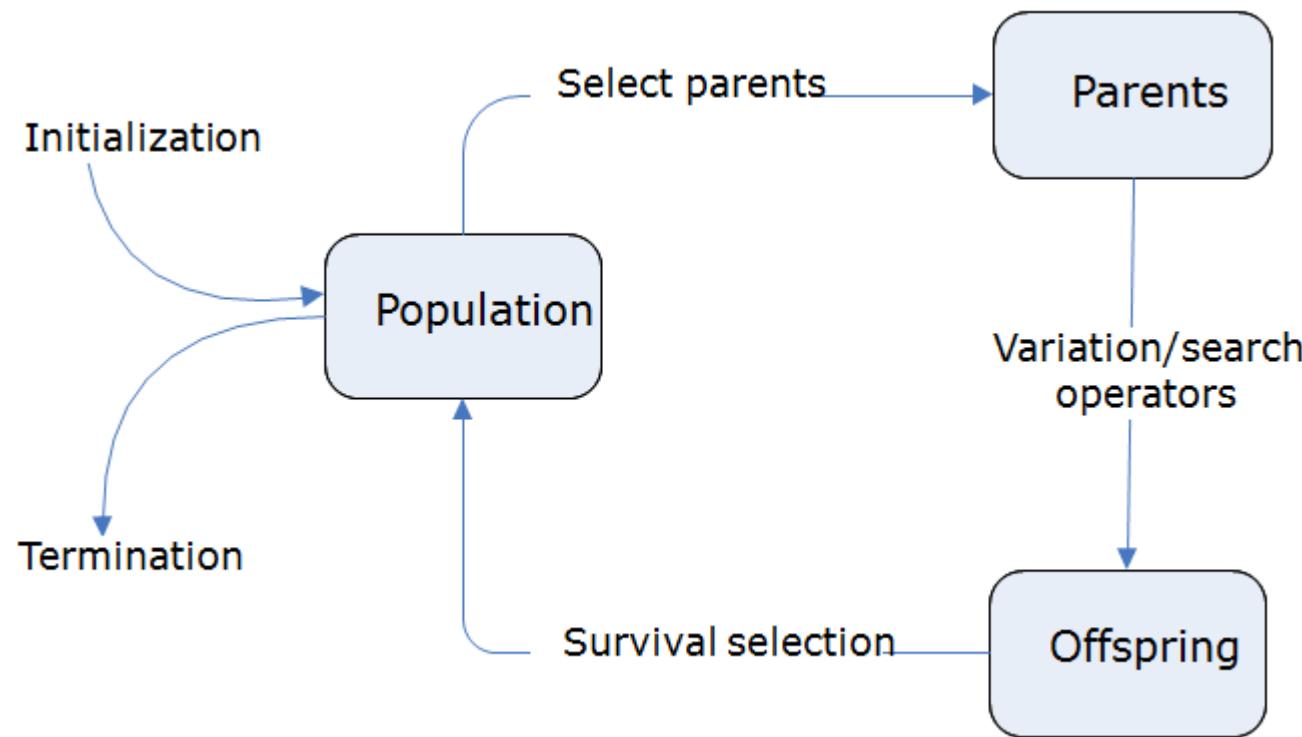
**Calitate** – sansa de a afecta noi solutii

# Istoric

- Inventati independent de 10 ori de diferiti cercetatori, cu proceduri usor diferite si nume diferite
- J. Holland (1975, SUA) – GA cunoscut azi sub numele de *Simple Genetic Algorithm (SGA)*
- J. Holland, K. DeJong, D. Goldberg
- Caracteristici
  - In mod traditional, informatii din parinti buni sunt refolosite (incrucisare)
  - Optimizare discreta
  - Probleme combinatoriale
- GAs difera prin:
  - Reprezentare
  - Mutatie, incrucisare
  - Mecanismul de selectie

Algoritmi Evolutivi

# Schema AE



# SAT – o abordare evolutiva

SAT cu 100 de variabile

$$F(x) = (x_{17} \vee \bar{x}_{37} \vee x_{73}) \wedge (\bar{x}_{11} \vee \bar{x}_{56}) \wedge \dots \wedge (x_2 \vee x_{43} \vee \bar{x}_{77} \vee \bar{x}_{89} \vee \bar{x}_{97})$$

Trebuie sa gasim valorile  $x_i, i = 1 \dots 100$  astfel incat  $F(x) = \text{TRUE}$

**(1) Reprezentare:** un sir de biti (0-FALSE, 1-TRUE) de lungime 100

Ex. 101010....10

**(2) Populatia:**

- Cati indivizi? *Sa zicem 30*
- Cum generam populatia initiala? *Aleator cu sanse 50-50 pentru fiecare bit sa fie 0 sau 1 ( $\Rightarrow$  diversitate)*

**(3) Evaluare:**

- Fiecare individ din populatie trebuie evaluat
- Daca  $F(x)=1$  am gasit solutia. Dar daca  $F(x)=0$  pentru toti indivizii?

# SAT – o abordare evolutiva

SAT cu 100 de variabile

$$F(x) = (x_{17} \vee \bar{x}_{37} \vee x_{73}) \wedge (\bar{x}_{11} \vee \bar{x}_{56}) \wedge \dots \wedge (x_2 \vee x_{43} \vee \bar{x}_{77} \vee \bar{x}_{89} \vee \bar{x}_{97})$$

Trebuie sa gasim valorile  $x_i, i = 1 \dots 100$  astfel incat  $F(x) = \text{TRUE}$

## (3) Functia de evaluare:

- Numarul de subexpresii (separate de  $\wedge$ ) din F evaluate la TRUE
- Sa presupunem ca exista in F 20 de asemenea subexpresii  $\Rightarrow$  cel mai bun fitness este 20 si cel mai slab este 0
- Sa presupunem ca cel mai bun individ are un fitness de 20, cel mai slab de 1 si media este intre 4 si 5

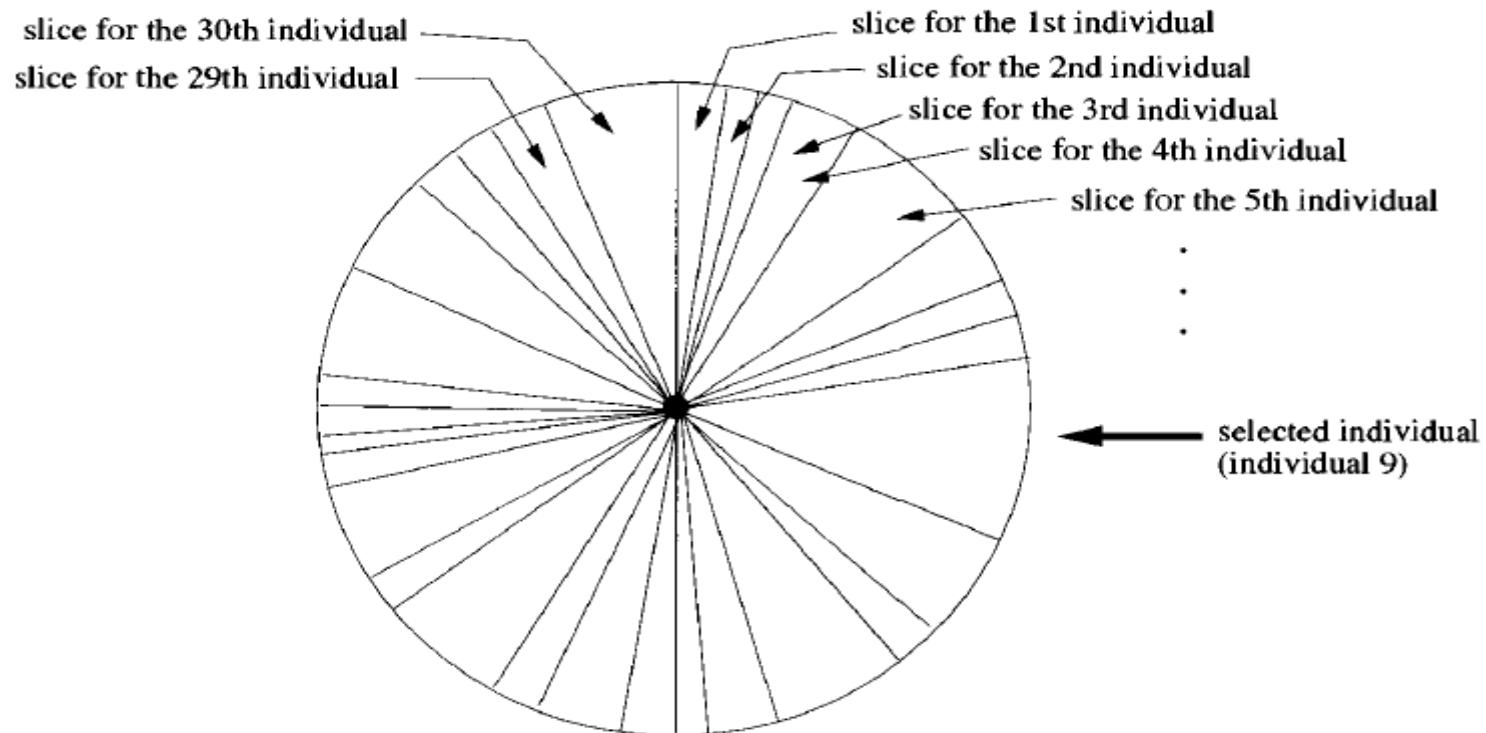
## (4) Selectia:

- Cum alegem indivizii care supravietuiesc? O metoda este **selectia proporcionala** cu fitnessul (*e.g. o solutie cu scor 10 are de 10 ori mai multe sanse sa fie aleasa decat una cu scor 1*)

# SAT – o abordare evolutiva

## (4) Selectia proporcionala (sau roulette wheel selection)

- Daca vrem sa avem tot 30 de indivizi in populatie, atunci am imparti ruleta in 30 de felii (una pentru fiecare individ) de marime proportionala cu fitnessul si am invarti ruleta de 30 de ori

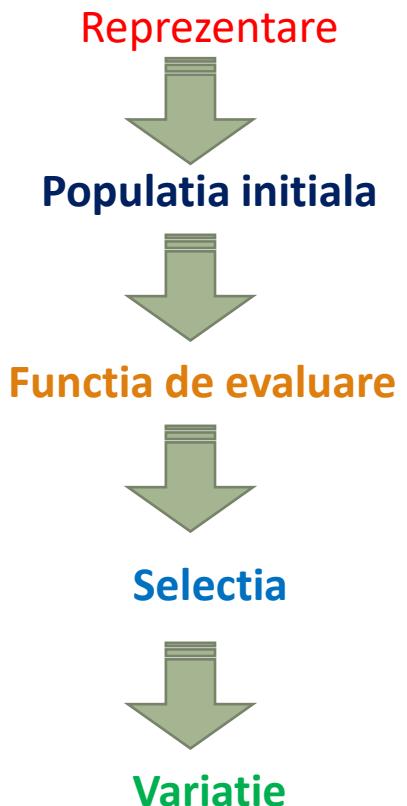


# SAT – o abordare evolutiva

- Pana acum indivizii din populatie nu au fost modificati

## (5) Variatie pentru a cauta noi solutii

- Fiecare individ poate sa aiba probabilitatea de 0.9 de intra in process de recombinare cu un alt individ ales aleator din cei 29 ramasi
  - Cut-and-splice procedure
- ✓ Asigura pastrarea unor secente din parinti care au generat pana acum solutii bune
- Poate insa genera solutii noi numai daca exista o diversitate suficienta in populatie



# TSP– o abordare evolutiva

TSP cu 100 de orase

(1) Reprezentare: permutare de 100

Ex. [14, 1, 43, 4, 6, 79, ..., 100, 2]

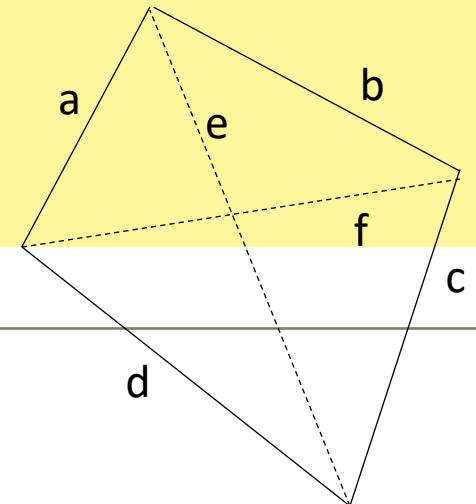
(2) Populatia:

- 30 indivizi
- Cum generam populatia initiala? *Permutari aleator generate.*

(3) Evaluare:

- Distanța totală de parcurgere a rutei data de permutare
- Cu cat distanță este mai mică, cu atât solutia este mai bună

# TSP – o abordare evolutiva



## (4) Variatie:

- Cum generam solutii noi din indivizii existenti?  
✓ Tinem cont de problema:

Care este mai mare: suma diagonalelor ( $e+f$ ) unui patrulater sau suma a două laturi opuse ( $a+c$  sau  $b+d$ )?

$e+f$  este întotdeauna mai mare  $\Rightarrow$  rute care se intersectează sunt întotdeauna mai lungi decât cele care nu se intersectează

- **Operator: alegem 2 orase din lista și inversăm ordinea în care apar**  
✓ Dacă era o rută ce se intersecta între ele, atunci am reparat soluția  
✓ Altfel, e posibil să generăm rute noi care se intersectează acum!
- **Posibilități de operatori care consideră 2 parinti?**

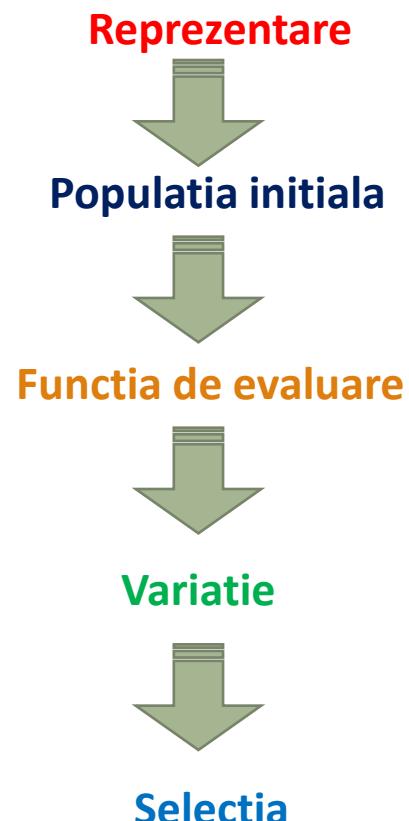
# TSP – o abordare evolutiva

## (5) Selectie:

- Sa zicem ca pentru fiecare parinte am generat 3 offspring
- Populatie = 30, Copii = 90

Ce putem face acum in faza de selectie?

1. Punem cei 90 de copii in competitie cu cei 30 de parinti si selectam o noua populatie din 120 de indivizi
  2. Pastram 30 de indivizi dintre cei 90 de copii si eliminam parintii
- ✓ A doua varianta - numita selectia  $(\mu, \lambda)$  unde  $\mu$  este numarul de parinti si  $\lambda$  este numarul de parinti – selectie (30,90)



# NLP– o abordare evolutiva

NLP cu 10 variabile

**(1) Reprezentare:** vector de 10 numere reale

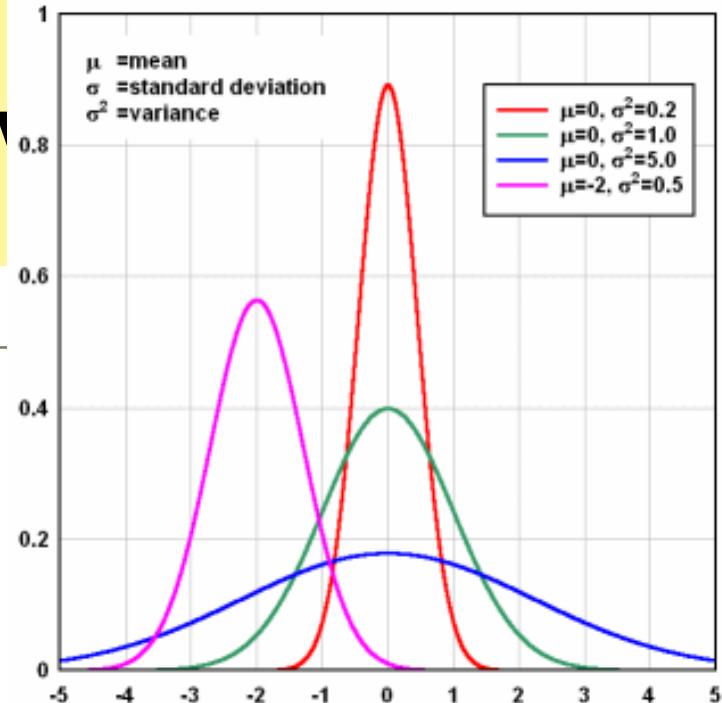
**(2) Populatia:**

- 30 indivizi
- Initializare
  - Generam aleator un numar real din intervalul  $[0,10]$  pentru fiecare variabila
  - ✓ Acest vector initial poate fi generat aleator pana cand unul va satisface toate restrictiile

# NLP– o abordare evolutivă

## (3) Variatie

- Adaugam o variabila gaussiana aleatoare fiecarei valori reale din solutie
- Standard Gaussian
  - ✓ Mean 0
  - ✓ Variance 1
- Fiecare offspring va fi similar cu parintele (10 schimbari, fiecare variabila fiind modificația usor prin adăugarea valorii Gaussiene aleator generate)
- Dacă variația este prea mare, legatura parinte-copil este ruptă ceea ce poate conduce la o căutare aleatoare care nu mai tine cont de ceea ce s-a învățat deja în procesul evolutiv



# NLP– o abordare evolutiva

## (4) Evaluare:

- Putem folosi direct functia de optimizat
- Acest obiectiv este definit peste toate numerele reale din spatiu
- Ce facem cu restrictiile care poate nu sunt satisfacute de individ?
  - Aplicam o penalizare functiei de fitness care depinde de cat este de departe individul curent de zona acceptata

# NLP– o abordare evolutiva

## (5) Selectia:

- Ranking selection using tournament
- Am inceput cu o populatie de 30 si am mai generat 30 de copii prin variatia gaussiana (1 offspring pentru 1 parinte)
- Selectam cate 2 indivizi aleator din cei 60; din fiecare pereche, alegem individul cu un fitness mai bun



# Un algoritm evolutiv standard

```
BEGIN
    INITIALISE population with random candidate solutions;
    EVALUATE each candidate;
    REPEAT UNTIL ( TERMINATION CONDITION is satisfied ) DO
        1 SELECT parents;
        2 RECOMBINE pairs of parents;
        3 MUTATE the resulting offspring;
        4 EVALUATE new candidates;
        5 SELECT individuals for the next generation;
    OD
END
```

# Decizii importante

- **Reprezentarea** indivizilor – codificare?
- Stabilirea functiei de **fitness** – cum este evaluat fiecare individ?
- **Operatori** de variație (cautare)



Orice model de rezolvare a unei probleme:  
+ **obiectiv** (poate diferi de functia de fitness)

# Populatia

- Posibile solutii
- De obicei numarul de indivizi este fix
- Operatorii de selectie actioneaza asupra intregii populatii curente
- ***Diversitatea*** unei populatii
  - Numarul de valori fitness/fenotipuri/genotipuri ***diferite*** din populatie

# Selectie

- **Mecanismul de selectie al parintilor**

- Fiecarui individ i se asociaza o anumita probabilitate de a deveni parinte (pe baza fitness-ului)
- Model probabilistic (des intalnit)
  - Solutiile bune (fitness bun) au sanse mai mari de a deveni parinti decat solutiile slabe
  - Totusi chiar si cel mai slab individ din populatie are o probabilitate mai mare ca zero de a deveni parinte

- **Selectia populatiei (survival selection/replacement)**

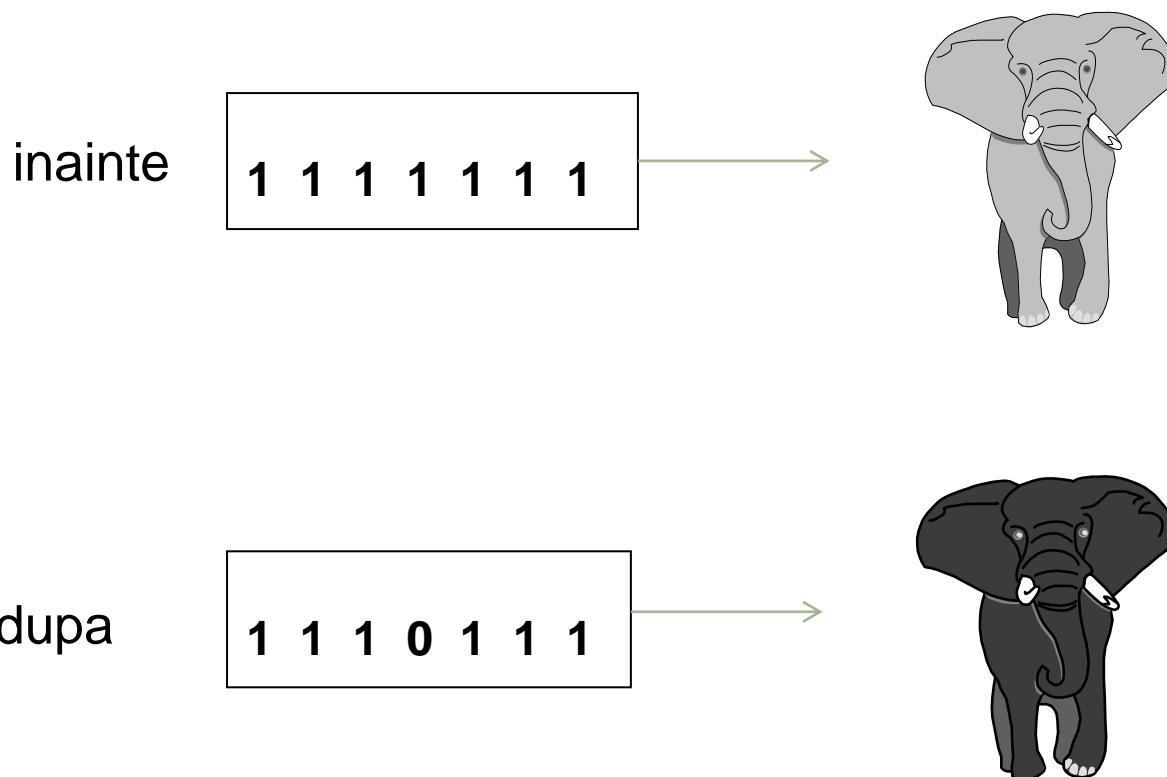
- Ce indivizi supravietuiesc in generatie urmatoare?
- Determinist: sorteaza parinti si copii si alege pe cei mai buni (pe baza fitness-ului), SAU genereaza copii si sterge toti parintii
- Combinatii (elitism)

# Operatori de variație

- Scop: generarea unor soluții noi
- Aritate = 1
  - **Mutatie (Mutation)**
- Aritate > 1
  - **Recombinare**
  - Aritate = 2 ⇔ **Incrucisare (Crossover)**
- Debate: mutatie sau incrusicare?
  - Amandouă
  - Definirea unor operatori specifici depinde de reprezentare

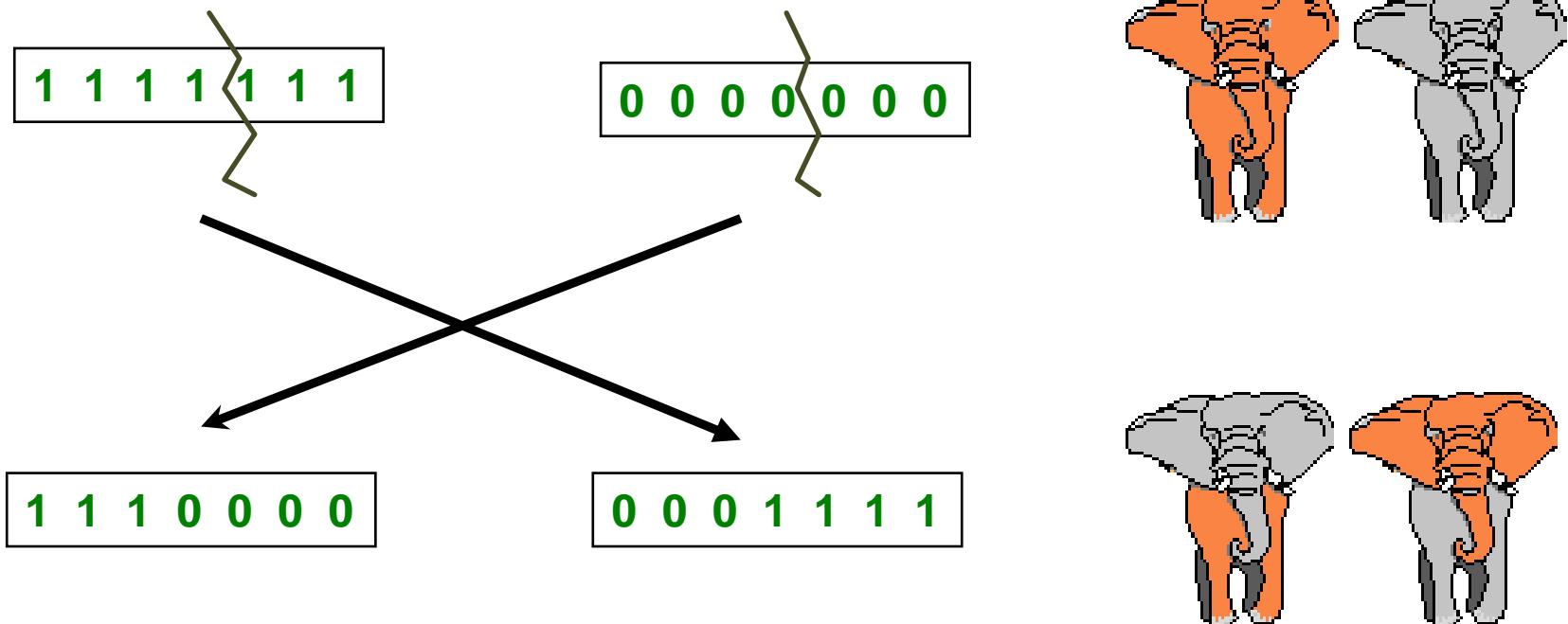
# Mutatie

- Produce o variație mică, aleatoare asupra unui individ și returnează un nou individ



# Recombinare

- Combina informatii din parinti in descendenti



# Start-Stop

- **Initializare**
  - De obicei **aleatoare**
  - Trebuie sa asigure o buna distributie a cromozomilor
  - Pot fi folosite euristici specifice problemei pentru determinarea unor solutii initiale
- **Criteriul de terminare**
  - Se verifica in fiecare generatie
  - Atingerea unui ***fitness*** (cunoscut sau presetat)
  - Atingerea unui ***numar maxim de generatii***
  - Atingerea unui grad minim de diversitate
  - Atingerea unui numar specificat de generatii in care fitness-ul nu a mai fost imbunatatit

# Simple GA (SGA) - Holland

- *Reprezentare*: binara
- *Recombinare*: N-point, uniforma
- *Mutatie*: bit flip cu probabilitate fixa
- *Selectia parintilor*: propotionala cu fitness
- *Selectia supravietuitilor*: toti urmasii inlocuiesc parintii
- Focus: incrucisare

# SGA: Reprezentare

- Codificare binara

- $\{0,1\}^L$

- 10010001

- 10010010

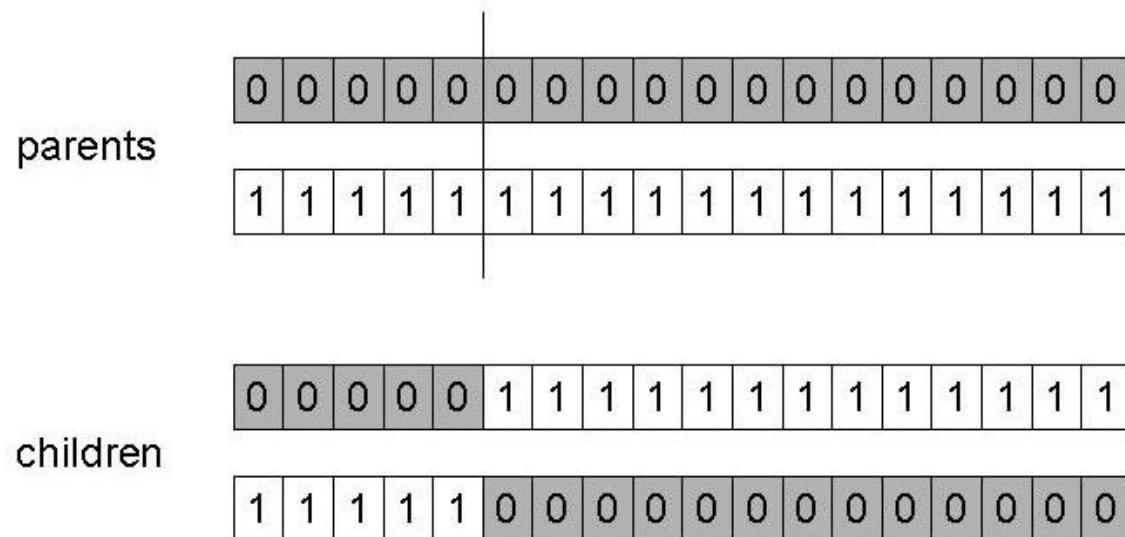
- 01110100

# SGA: Schema

1. Selectia parintilor din ‘mating pool’ (= populatie)
2. Schimbarea aleatoare a ordinii in mating pool
3. Pentru fiecare pereche consecutiva se aplica incrusicare cu probabilitatea  $p_c$  , altfel se copiaza parintii
4. Pentru fiecare offspring se aplica mutatie: pe fiecare bit cu probabilitatea  $p_m$
5. Inlocuirea intregii populatii cu noii indivizi creati (urmasii - offspring)

# SGA: Incrucisare cu un punct de taietura

- Se alege aleator un punct de taietura pentru parinti
- Se creaza urmasi prin interschimbarea partilor
- Probabilitatea  $p_c$  - de obicei in  $(0.6, 0.9)$



# SGA: Mutatie

- Modifica fiecare pozitie din cromozom cu probabilitatea  $p_m$
- $p_m$  - rata de mutatie (de obicei  $\sim 0.2$ )

parent      

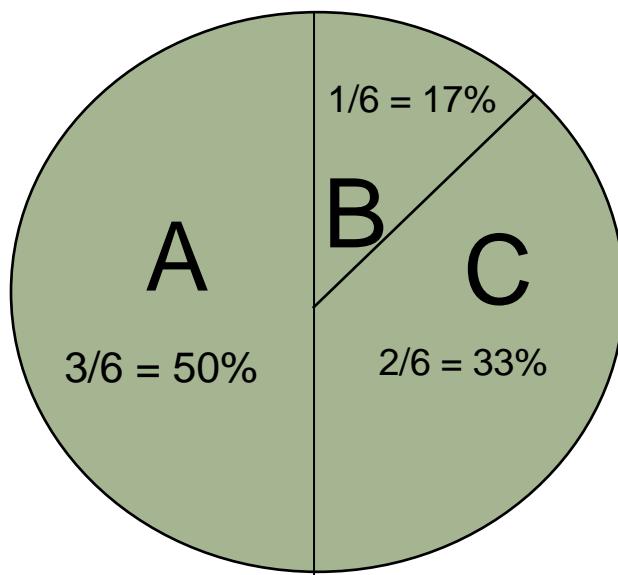
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

child      

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

# SGA: Selectia

- Indivizii mai buni au o sansa mai mare – proportionala cu fitness-ul
- Implementare: tehnica ‘roulette wheel’ (ruleta)
  - Fiecare individ primeste o felie din ruleta
  - Ruleta se invarte de n ori pentru a selecta n indivizi



$$\text{fitness}(A) = 3$$

$$\text{fitness}(B) = 1$$

$$\text{fitness}(C) = 2$$

# SGA: Remarci

- Folosit in comparatii cu algoritmi evolutivi noi
- Puncte slabe:
  - Reprezentarea este restrictiva
  - Mutatia si incruisarea sunt aplicabile numai in codificarea binara
  - Mecanismul de selectie inefficient cand este vorba de indivizi cu un fitness apropiat
  - Modelul de populatie generational poate fi imbunatatit prin alegerea explicita a supravietuitorilor

# Structura generala AE

**begin**

$t \leftarrow 0$

Initializare  $P(t)$

Evaluare  $P(t)$

**while** (not termination-condition) **do**

**begin**

$t \leftarrow t + 1$

Select  $P(t)$  din  $P(t-1)$

Modificare  $P(t)$

Evaluare  $P(t)$

**end**

**end**

Next time...

## More on AEs



**BABEŞ-BOLYAI UNIVERSITY**  
Faculty of Mathematics and Computer Science



# Inteligentă Artificială

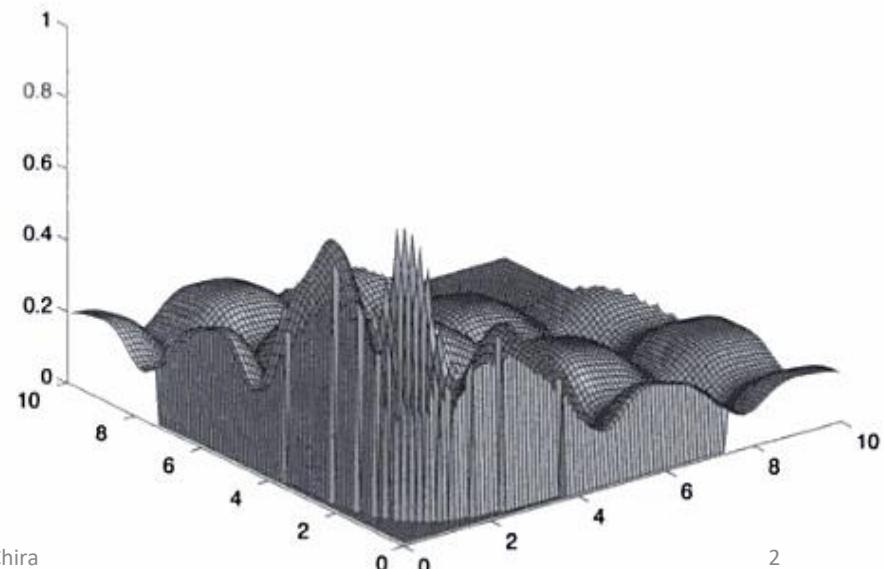
*5: Proiectarea Algoritmilor Evolutivi*

**Camelia Chira**

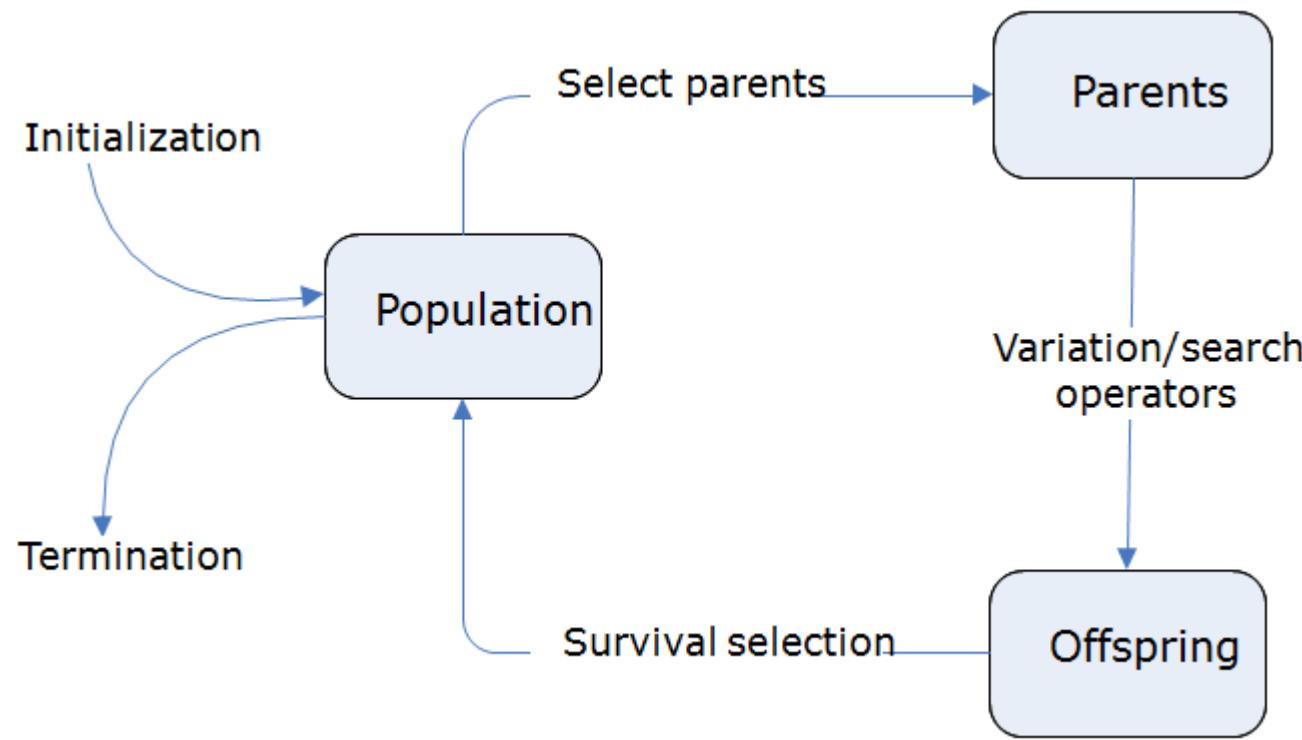
[cchira@cs.ubbcluj.ro](mailto:cchira@cs.ubbcluj.ro)

# Algoritmi evolutivi

- O *populatie* de solutii candidat este evaluata in iteratii succesive de *variatie si selectie aleatoare*.
  - Reprezentarea solutiilor
  - Functia criteriu (de evaluare a solutiilor)
  - Operatori specifici de variatie si selectie
  - Marimea si initializarea populatiei
  - *Optimum set of choices?*
- NU exista o metoda optima de a proiecta un algoritm de cautare eficient pentru orice problema!
- NU exista o singura metoda cea mai buna de a cauta solutia in orice problema individuala!



# Schema AE



# An example (after Goldberg)

- Simple problem:  $\max x^2$  over  $\{0,1,\dots,31\}$
- GA approach:
  - Representation: binary code, e.g.  $01101 \leftrightarrow 13$
  - Population size: 4
  - 1-point xover, bitwise mutation
  - Roulette wheel selection
  - Random initialisation

Sa vedem ce se intampla intr-o generatie...

## $\chi^2$ example: selection

String no.	Initial population	$x$ Value	Fitness $f(x) = x^2$	$Prob_i$	Expected count	Actual count
1	0 1 1 0 1	13	169	0.14	0.58	1
2	1 1 0 0 0	24	576	0.49	1.97	2
3	0 1 0 0 0	8	64	0.06	0.22	0
4	1 0 0 1 1	19	361	0.31	1.23	1
Sum			1170	1.00	4.00	4
Average			293	0.25	1.00	1
Max			576	0.49	1.97	2

## X<sup>2</sup> example: crossover

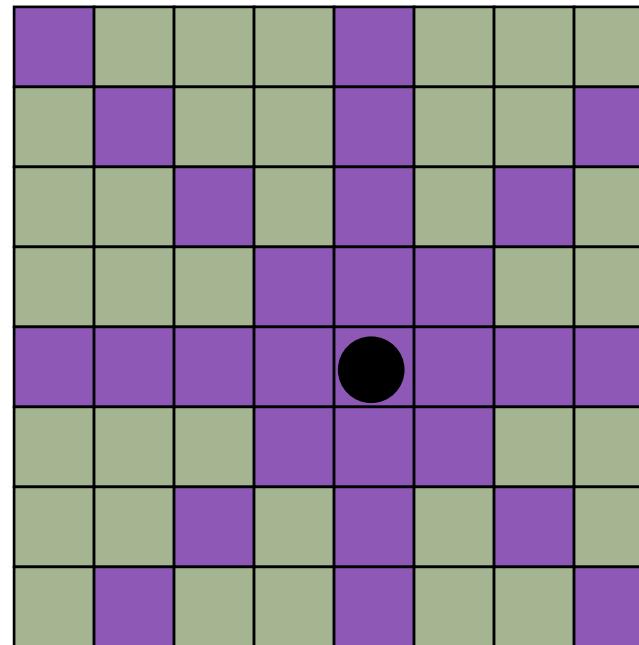
String no.	Mating pool	Crossover point	Offspring after xover	x Value	Fitness $f(x) = x^2$
1	0 1 1 0   1	4	0 1 1 0 0	12	144
2	1 1 0 0   0	4	1 1 0 0 1	25	625
2	1 1   0 0 0	2	1 1 0 1 1	27	729
4	1 0   0 1 1	2	1 0 0 0 0	16	256
Sum					1754
Average					439
Max					729

## $x^2$ example: mutation

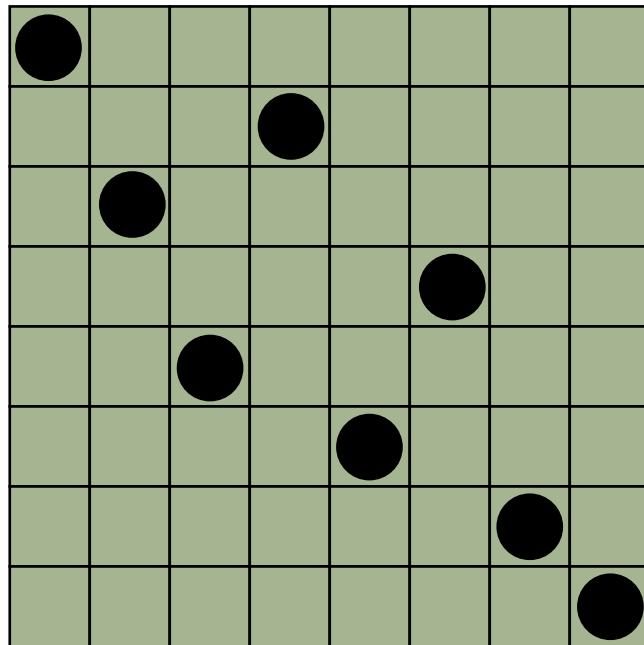
String no.	Offspring after xover	Offspring after mutation	$x$ Value	Fitness $f(x) = x^2$
1	0 1 1 0 0	1 1 1 0 0	26	676
2	1 1 0 0 1	1 1 0 0 1	25	625
2	1 1 0 1 1	1 1 0 1 1	27	729
4	1 0 0 0 0	1 0 1 0 0	18	324
Sum				2354
Average				588.5
Max				729

# 8Q: Problema celor 8 regine

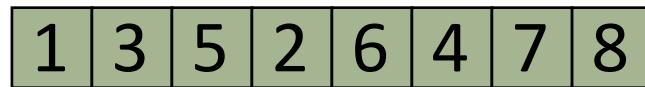
- Asezati 8 regine pe o tabla de sah 8x8 astfel incat sa nu se atace



## 8Q: Reprezentare



Phenotype



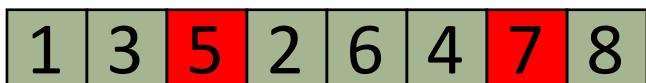
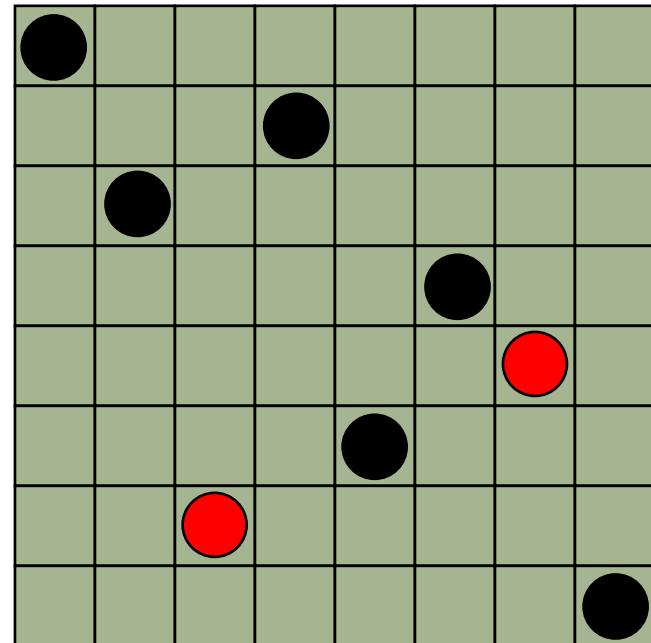
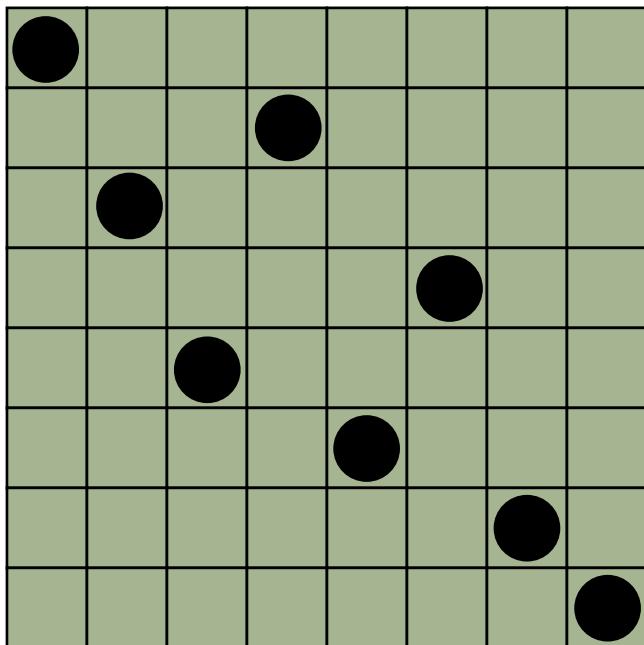
Genotype

## 8Q: Functia de fitness

- Penalizare
  - Pentru o regina – numarul de regine pe care le ataca
  - Pentru o configuratie – suma penalizarilor /regina
  - Scop: minimizarea penalizarii
  - Optim: 0
- Fitness-ul unei configuratii este inversul penalizarii (maximizare)

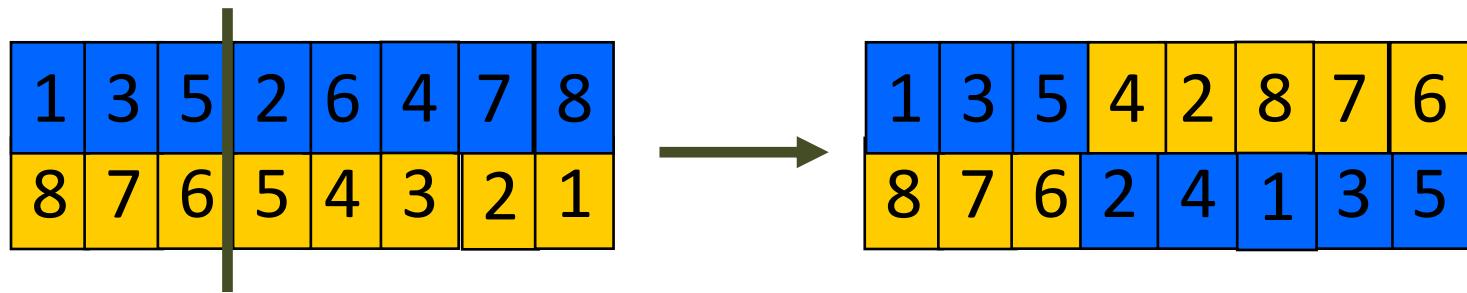
# 8Q: Mutatie

- Variatii in permutari
- Schimb 2 pozitii aleatoare (Swap)



## 8Q: Recombinare

- Combinarea a doua permutari pentru a obtine 2 permutari noi
- Exemplu
  - Aleg un punct de incrusicare aleator
  - Copiez prima parte de la fiecare parinte in cei doi copii
  - Construiesc a doua parte de la celalalt parinte



## 8Q: Selectie

- Selectia parintilor:
  - Alege  $k$  indivizi din populatia curenta si cei mai buni 2 – parinti – incrusisare
- Selectia supravietuitorilor:
  - Un nou cromozom creat (offspring) inlocuieste un individ din populatie
  - Varianta 1:
    - Sortarea populatiei dupa fitness
    - Primul individ cu un fitness mai slab decat noul individ este inlocuit
  - Varianta 2 (des folosita):
    - Inlocuieste cel mai slab individ din intreaga populatie

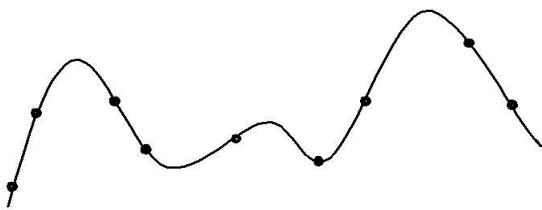
## 8Q: Parametri

Populatie - size	100
Initializare	Aleatoare
Probabilitate de recombinare	100%
Recombinare	“Cut-and-crossfill” crossover
Selectia parintilor	Cei mai buni 2 din k = 10 aleatori
Probabilitate de mutatie	80%
Mutatie	Swap
Selectia supravietuitorilor	Replace worst
Conditie de stop	Solutie gasita sau 10,000 de evaluari fitness

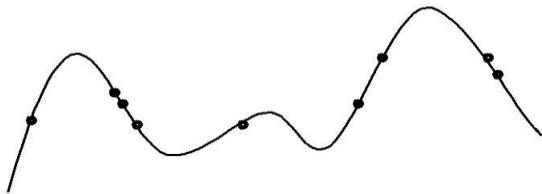
# AEs

- AE permit folosirea oricarei reprezentari dar operatorii de variație vor depinde de reprezentarea aleasă
- Există multe diferențe între AEs, însă toți au cam aceeași “retetă” de evaluare-selectie-variație:
  - Crearea unei populații de indivizi ce reprezintă soluții potențiale
  - Evaluarea indivizilor
  - Introducerea unei presiuni de selecție ce promovează indivizii mai buni și îi elimină pe cei mai slabii
  - Aplicarea unor operatori de variație pentru a genera soluții noi
- AEs pot combina cele 2 categorii de algoritmi (soluții complete vs incomplete): indivizii pot descrie spații suplimentare sau soluții particulare
- Parametri
  - Există: marimea populației, probabilități de încrucișare/mutare
  - Pot fi însă evaluați: *Tuning the algorithm to the problem while solving the problem!*

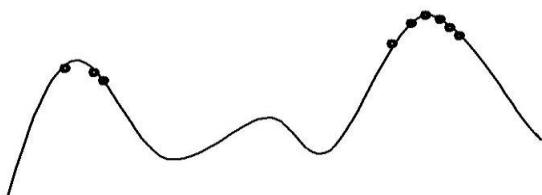
# EA: comportament tipic



**Inceputul rularii:**  
Distributia populatiei cvasi-aleator



**Mijloc:**  
Populatia se aranjeaza spre dealuri



**Final:**  
Populatia se concentreaza de dealurile inalte

# Avantaje AEs

- Inerent adaptivi
  - Individii se pot adapta natural la schimbari din mediu – probleme dinamice
  - AEs nu trebuie restartati cu fiecare schimbare in model
- Usor de hibridizat cu alte tehnici
  - Operatori de variatie specializati: ex. hill-climbing
- Modele co-evolutive
- Paralelism
  - Ex. Un procesor opereaza cu un individ sau cu o subpopulatie

*...you are playing the role the creator!*

# Decizii importante in proiectarea AE

- **Reprezentarea** indivizilor
- **Functia de evaluare** sau de **fitness** – cum este evaluat fiecare individ?
- Operatori de **variatie**
- **Selectia**
- **Initializarea**



# Initializarea

- EA inseamna:

$$x[t + 1] = s(v(x[t]))$$

unde

- $x[t]$  este populatia cu reprezentarea  $x$  la timpul  $t$
- $v$  reprezinta operatorii de variatie
- $s$  este operatorul de selectie
- Populatia initiala  $x[0]$  trebuie determinata *inainte* de a incepe procesul evolutiv
- *Tipic: populatia initiala este generata aleator*
- Alte metode?
  - Putem tine cont de problema
  - Putem initializa un individ cu cea mai buna solutie gasita de un alt algoritm (ex. greedy)
- Diversitatea populatiei

# Reprezentare

- O reprezentare buna permite aplicarea unor operatori de cautare care sa mentina o legatura functionala intre parinti si copii
  - Nepotrivirea operatorilor de cautare si a reprezentarii => cautare aleatoare (sau chiar mai rau)
  - Recomandare: alegeti intotdeauna o reprezentare care este intuitiva pentru problema data
    - Codificarea binara
    - Codificarea reala
    - Codificarea specifica
- Vectori de lungime fixa
  - Permutari
  - Expresii simbolice

# Reprezentare: Vectori de lungime fixa

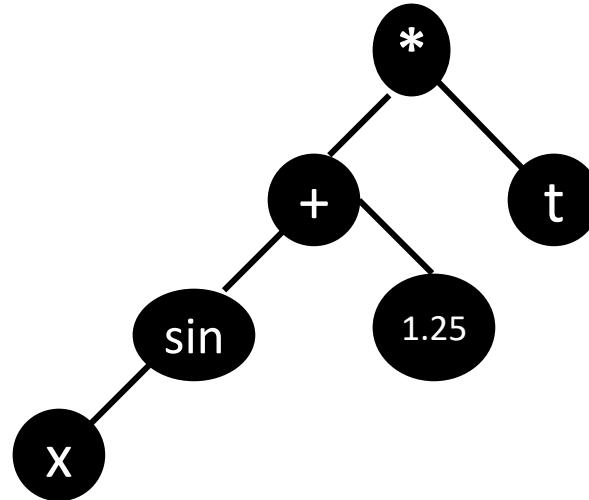
- ***Subset selection problem***
  - Obiectiv: ce obiecte vor fi alese dintr-o multime
  - Aplicatii: statistica, optimizare in transporturi
  - Reprezentare: lista de biti {0,1}
- Optimizarea unghiurilor de torsiune intr-un compus chimic
  - Vector de valori reale in  $[0, 2\pi)$
- ***Model liniar de predictie a unor valori***
  - Estimarea unui pret pentru o actiune se bazeaza pe istoric
  - $y[t] = a_1y[t - 1] + \dots + a_ky[t - k]$ , unde  $y[i]$  este sirul de preturi din istoric si  $a$  sunt coeficientii
  - Reprezentare:  $[k, a_1, \dots, a_{max}]$ , unde  $max$  este numarul maxim de valori precedente considerate
  - Operatorii de cautare pentru  $k$  difera de cei pentru  $a$

# Reprezentare: Permutari

- Problema de planificare a sarcinilor: *o lista de  $j$  sarcini trebuie ordonata pentru a fi executate intr-o fabrica*
  - Ordinea pozitiilor - importanta
- TSP
  - Pozitiile *adiacente* - importante
- **Reprezentare: [1,2,...,j]**
  - Ordinea din permutare corespunde ordinii aplicarii sarcinilor
  - E posibila existenta a mai multor instante pentru anumite sarcini

# Reprezentare: Arbori

- Probleme ce necesita construirea unei functii (pentru o anumita sarcina de mapare)
- Reprezentare
  - Expresie simbolica in forma unui arbore
  - $(*(+\sin(x) 1.25) (t))$
  - $(\sin(x)+1.25)*t$



# Functia de evaluare

- Este modul de a determina calitatea solutiilor evaluate
- Common sense: “*The optimum solution(s) should be given the optimum evaluation(s)*”.

## **Problema One Max**

*Obiectiv: gasiti un vector de 10 valori din {0,1} astfel incat numarul de 1 este maximizat.*

*Evident: best solution is [1111111111]*

$$f(x) = \sum_{i=1}^n x_i$$

*vs.*

$$f(x) = \prod_{i=1}^n x_i$$

- Scopul este de a obtine solutii bune intr-un timp scurt => functia de evaluare trebuie sa fie capabila sa ghideze cautarea si spre solutii mai putin perfecte
- **Evaluarea solutiilor consuma de obicei cel mai mult timp intr-un AE**

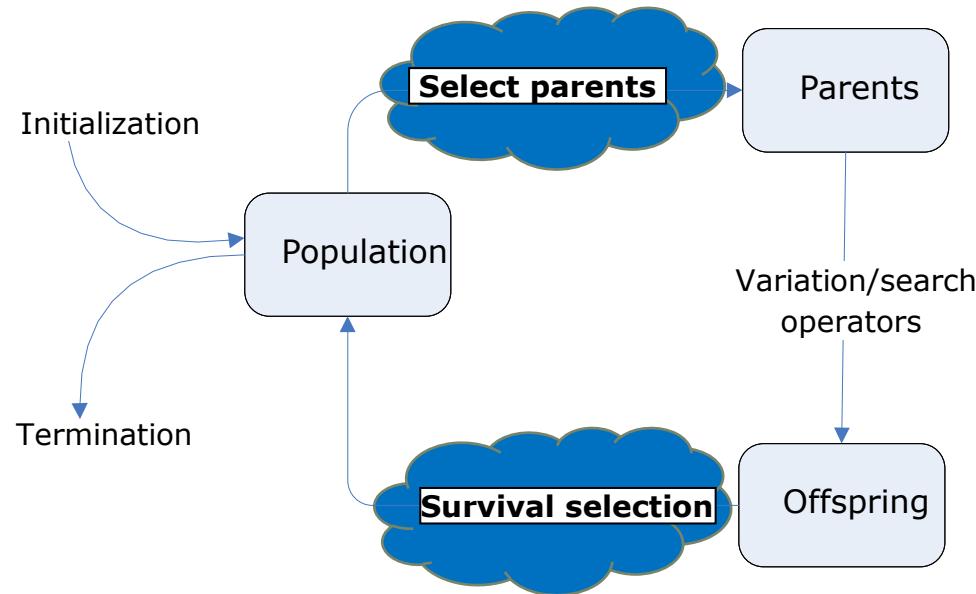
# Operatorii de cautare (variatie)

- De obicei se aleg pe baza reprezentarii alese
- Implementarea practica a operatorilor se bazeaza pe reprezentare si pe intutia despre cum arata spatiul de cautare (landscape) indus de functia de evaluare
- Orice spatiu de cautare are si o metrica de distanta: de obicei, numarul de operatori necesari pentru a trece dintr-un punct in altul
- Operatori pentru 1 parinte, 2 parinti, combinatii...
- **NU exista o cea mai buna alegere pentru toate problemele!**

# Selectia

Termenul poate fi folosit in 2 contexte:

- Selectia parintilor
- Selectia individizilor pentru generatia urmatoare



Putem avea tipuri diferite de selectie pentru cele 2 aspecte

Ex:

- Selectia proporcionala pentru a alege parintii
- Selectie turnir pentru a elimina inividui din populatie
- Regula elitista pentru a nu pierde cea mai buna solutie

# Selectia

- Determinarea noii populatii pe baza calitatii indivizilor
    - 1) Unii indivizi sunt eliminati in timp ce ceilalți supravietuiesc pentru a deveni parinti in generatia urmatoare
      - Un individ poate aparea o singura data in generatia urmatoare
    - 2) Indivizii sunt selectati pe baza calitatii lor relative
      - Un individ poate fi ales de mai multe ori
  - Filtrul prin care algoritmul stabileste generatia urmatoare
- 
- Determinista
    - Selectia elimina aceiasi indivizi
  - Stocastica
    - Supravietitorii sunt alesi in mod probabilistic

# Selectia parintilor

- Indivizii mai performanti au mai multe sanse de reproducere
- Stabileste indivizii din populatia curenta ce vor contribui la constituirea generatiei urmatoare
- Se bazeaza pe calitatea indivizilor (functia de fitness)
- Sarcina selectiei: ***exploatarea*** celor mai bune solutii gasite

# Selectia determinista

- $\mu$  – numarul parintilor
- $\lambda$  – numarul descendenților
- **Selectia ( $\mu + \lambda$ )**
  - $\lambda$  descendenti sunt creati din  $\mu$  parinti
  - $\mu + \lambda$  indivizi sunt evaluati
  - Selectia: Cei mai buni  $\mu$  sunt pastrati
- **Selectia ( $\mu, \lambda$ )**
  - $\lambda$  descendenti sunt creati din  $\mu$  parinti,  $\lambda \geq \mu$
  - Selectia: Cei mai buni  $\mu$  din cei  $\lambda$  indivizi sunt pastrati

# Selectia stocastica

- Selectia proporcională
- Selectia prin ordonare
- Selectia turnir
- Selectia elitista
- etc

# Selectia proporcională

$$P(t) = \{x_1, x_2, \dots, x_n\}$$

- Functia de fitness  $f: X \rightarrow R; f(x) \geq 0$  (scalare altfel)

- Performanta totala a populatiei  $F = \sum_{i=1}^n f(x_i)$

- Probabilitatea de selectie pentru  $x_i$

$$p_i = \frac{f(x_i)}{F}$$

- Selectia se aplica de  $n$  ori  $\Rightarrow$  valoarea medie a descendantilor individului i este:

$$n_i = n \cdot p_i = \frac{n \cdot f(x_i)}{\sum_{i=1}^n f(x_i)} = \frac{f(x_i)}{\bar{f}}$$

# Algoritmul Monte Carlo de selectie (Algoritmul ruletei)

P1. Pentru fiecare cromozom se calculeaza:

$$q_i = \sum_{k=1}^i p_k; i = 1, 2, \dots, n$$

P2. Pentru  $i=1,2,\dots,n$  executa

P2.1 Se genereaza aleator  $g$  in intervalul  $[0,1]$

P2.2 Regula de selectie:

- Daca  $0 \leq g \leq q_1$  atunci este selectat  $x_1$
- Daca  $q_{i-1} \leq g \leq q_i$  atunci este selectat  $x_i$

- Dominanta unui individ performant => convergenta prematura!
- Spre sfarsitul rularii – fitness similar => se pierde presiunea de selectie

# Selectia bazata pe ordonare (Rank based)

- Aranjarea indivizilor in ordinea crescatoare a calitatii (maximizarea functiei de fitness): worst primeste rank 1, fittest primeste rank n
- *Probabilitatea de selectie a unui individ depinde de rangul sau in sirul ordonat => se foloseste un fitness relativ nu unul absolut!*
- Presiunea de selectie - numarul mediu de descendenti ai celui mai performant individ
  - Constanta a metodei
  - Se exprima printr-un numar s din [1,2]

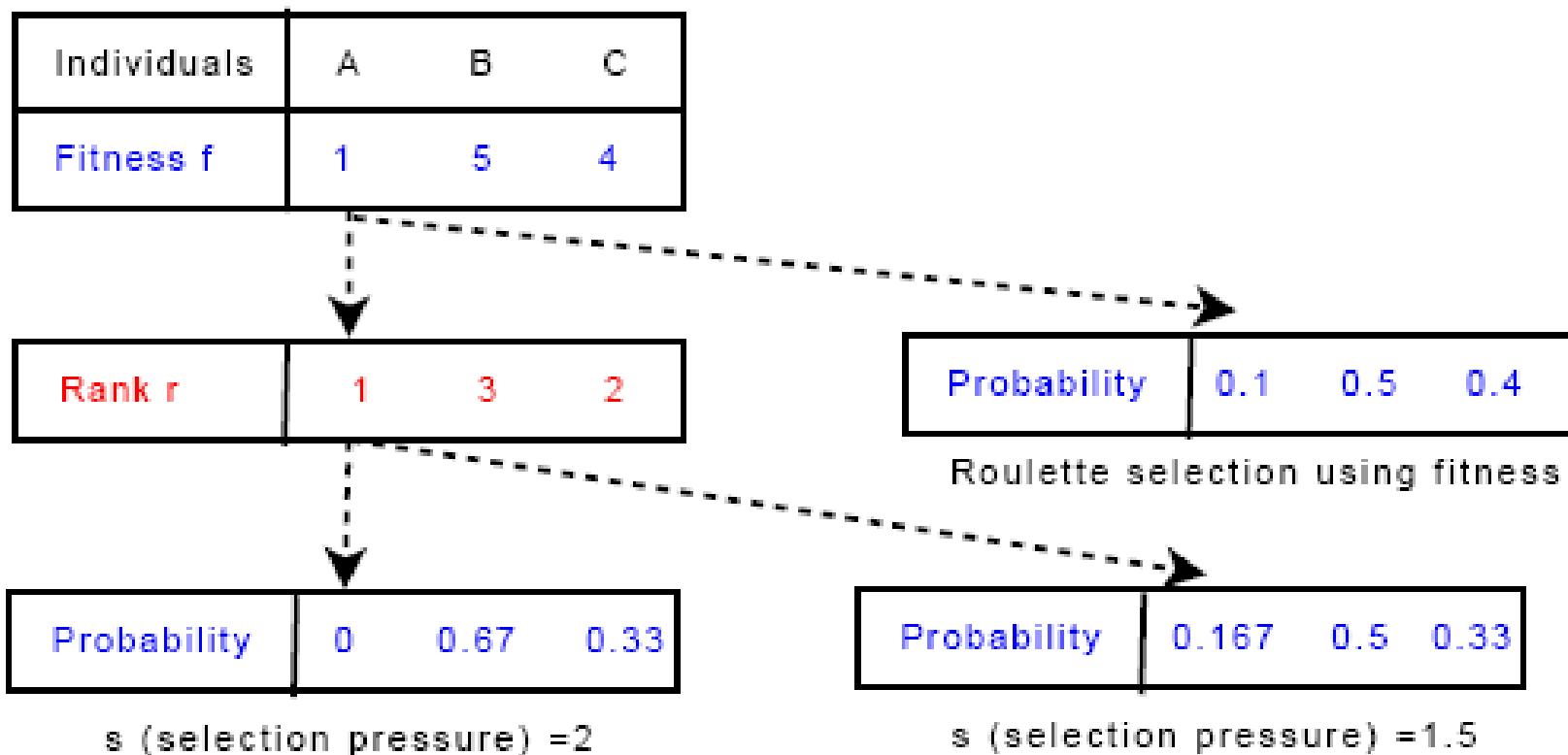
# Selectia prin ordonare

$$P(t) = \{x_1, x_2, \dots, x_n\}$$

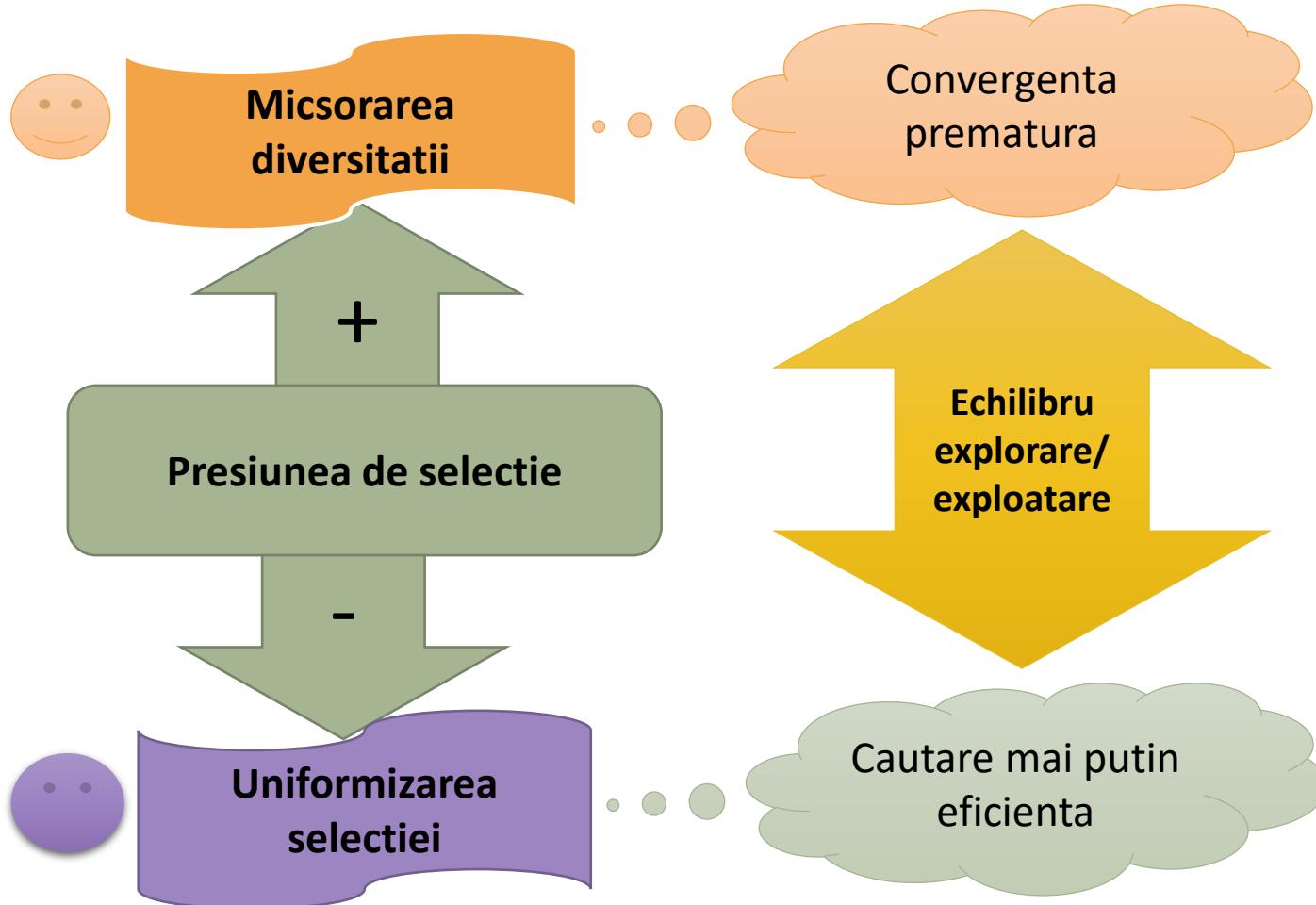
- $r_i$  – rangul individului i ( $r_i = 1$  pentru cel mai slab individ)
- $s$  – presiunea de selectie
- Probabilitatea de selectie depinde de pozitia relativa a cromozomilor:

$$p_i = \frac{2 - s}{n} + \frac{2(r_i - 1)(s - 1)}{n - 1}$$

# Rank based selection: Exemplu

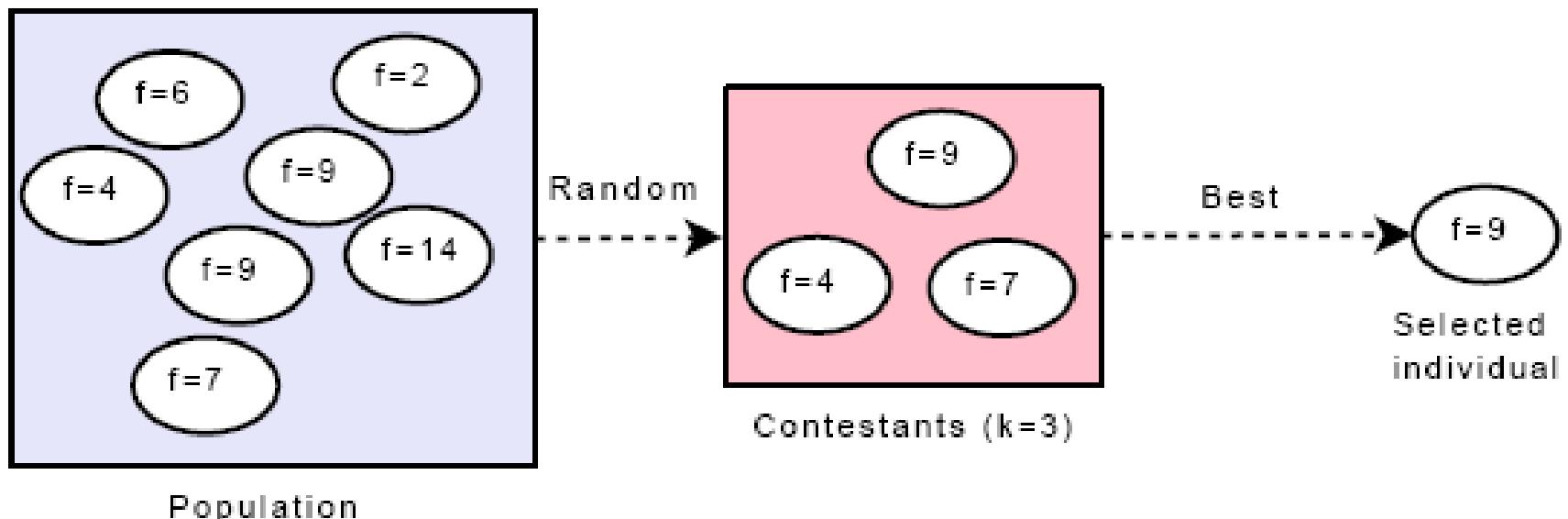


# Observatii



# Selectia turnir (tournament size k)

- Se aleg in mod aleator k cromozomi
- Se calculeaza fitness-ul cromozomilor selectati
- Cromozomul mai performant este selectat
- Mecanismul este aplicat de n ori pentru a selecta n parinti



# Alte mecanisme de selectie

- Strategii elitiste
- Folosirea unei rate de inlocuire
  - Numai o parte din parinti sunt inlocuiti
  - Cromozomii noi generati inlocuiesc indivizii apropiati de ei
- Inlocuirea asincrona a indivizilor (vs generationala)
- Selectie dinamica (vs statica)

# Structura generala AE

**begin**

$t \leftarrow 0$

Initializare  $P(t)$

Evaluare  $P(t)$

**while** (not termination-condition) **do**

**begin**

$t \leftarrow t + 1$

Select  $P(t)$  din  $P(t-1)$

Modificare  $P(t)$

Evaluare  $P(t)$

**end**

**end**

# Generational GA

```
Initialization P(0)
Evaluation(P(0))
g = 0;
While (not stop_condition) do
    Repeat
        Select 2 parents p1 and p2 from P(g)
        Crossover(p1,p2) =>o1 and o2
        Mutation(o1) => o1*
        Mutation(o2) => o2*
        Evaluation(o1*)
        Evaluation(o2*)
        Add o1* and o2* into P(g+1)
    Until P(g+1) is full.
    g++
EndWhile
```

# Steady-state GA

```
Initialization P
Evaluation(P)

While (not stop_condition) do
    Select 2 parents p1 and p2 from P
    Crossover(p1,p2) =>o1 and o2
    Mutation(o1) => o1*
    Mutation(o2) => o2*
    Evaluation(o1*)
    Evaluation(o2*)
    Best(o1*,o2*) replaces Worst(P)

EndWhile
```

Next time...

## More on AEs



**BABEŞ-BOLYAI UNIVERSITY**  
Faculty of Mathematics and Computer Science



# Inteligentă Artificială

*6: Codificarea binara, reala, permutari*

Camelia Chira

[cchira@cs.ubbcluj.ro](mailto:cchira@cs.ubbcluj.ro)

# Important: modalitatea de examinare IA

## Examen: P2 =max.550 puncte

1. Alegeti o tema de cercetare din domeniul IA.
2. Cautati un articol recent (publicat in ultimii 10 ani) care prezinta o metoda/algoritm in abordarea temei alese.
3. Reproduceti rezultatele din articolul ales.
4. Propuneti cel putin o modificare a metodei din articol si comparati rezultatele.
5. Scrieti un raport care sa descrie problema, metodele, experimentele, rezultatele si analiza lor.

### Evaluare:

- Prezentare = 250p (de submis pana in 9 mai, programarea prezentarilor 10 si 17 mai)
- Raport si cod sursa = 300p (de submis pana la data examenului)

# Important: modalitatea de examinare IA

## Examen: P2 =max.550 puncte

### Evaluare:

- **Prezentare = 250p**

- De submis pana in 9 mai, programarea prezentarilor 10 si 17 mai
- Trebuie sa contine toate rezultatele si analiza lor
- Maxim 10 slide-uri
- 5 minute

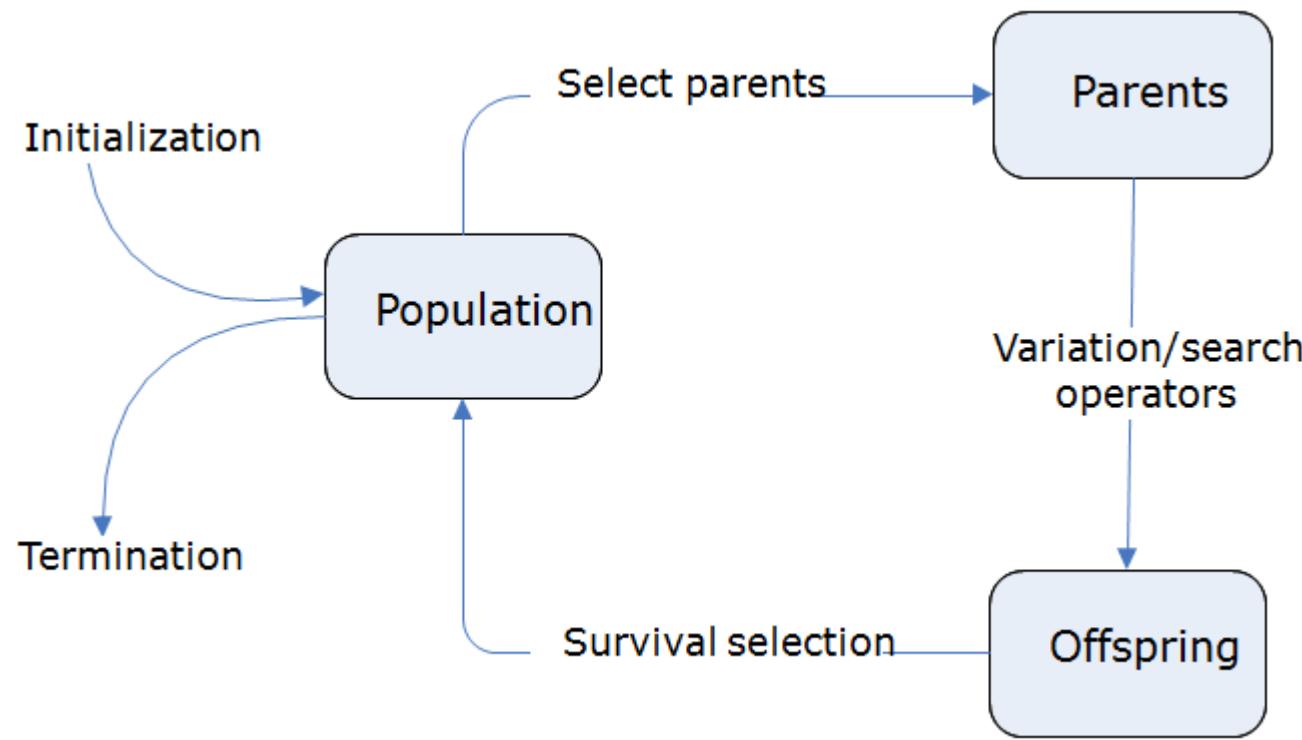
- **Raport (insotit de codul sursa) = 300p**

- De submis pana la data examenului
- Structura: introducere, prezentarea articolului de pornire, rezultatele replicate, modificarea propusa, analiza comparativa, concluzii
- Maxim 7 pagini

# Recap: AEs

- AE permit folosirea oricarei reprezentari dar operatorii de variație vor depinde de reprezentarea aleasă
- Există multe diferențe între AEs, însă toți au cam aceeași “retetă” de evaluare-selectie-variație:
  - Crearea unei populații de indivizi ce reprezintă soluții potențiale
  - Evaluarea indivizilor
  - Introducerea unei presiuni de selecție ce promovează indivizii mai buni și îi elimină pe cei mai slabii
  - Aplicarea unor operatori de variație pentru a genera soluții noi
- AEs pot combina cele 2 categorii de algoritmi (soluții complete vs incomplete): indivizii pot descrie spații suplimentare sau soluții particulare
- Parametri
  - Există: marimea populației, probabilități de încrucișare/mutare
  - Pot fi însă evaluați: *Tuning the algorithm to the problem while solving the problem!*

# Schema AE



# Decizii importante in proiectarea AE

- **Reprezentarea** indivizilor
- **Functia de evaluare** sau de **fitness** – cum este evaluat fiecare individ?
- Operatori de **variatie**
- **Selectia**
- **Initializarea**

# **Codificarea binara**

**Incrucisare, Mutatie**

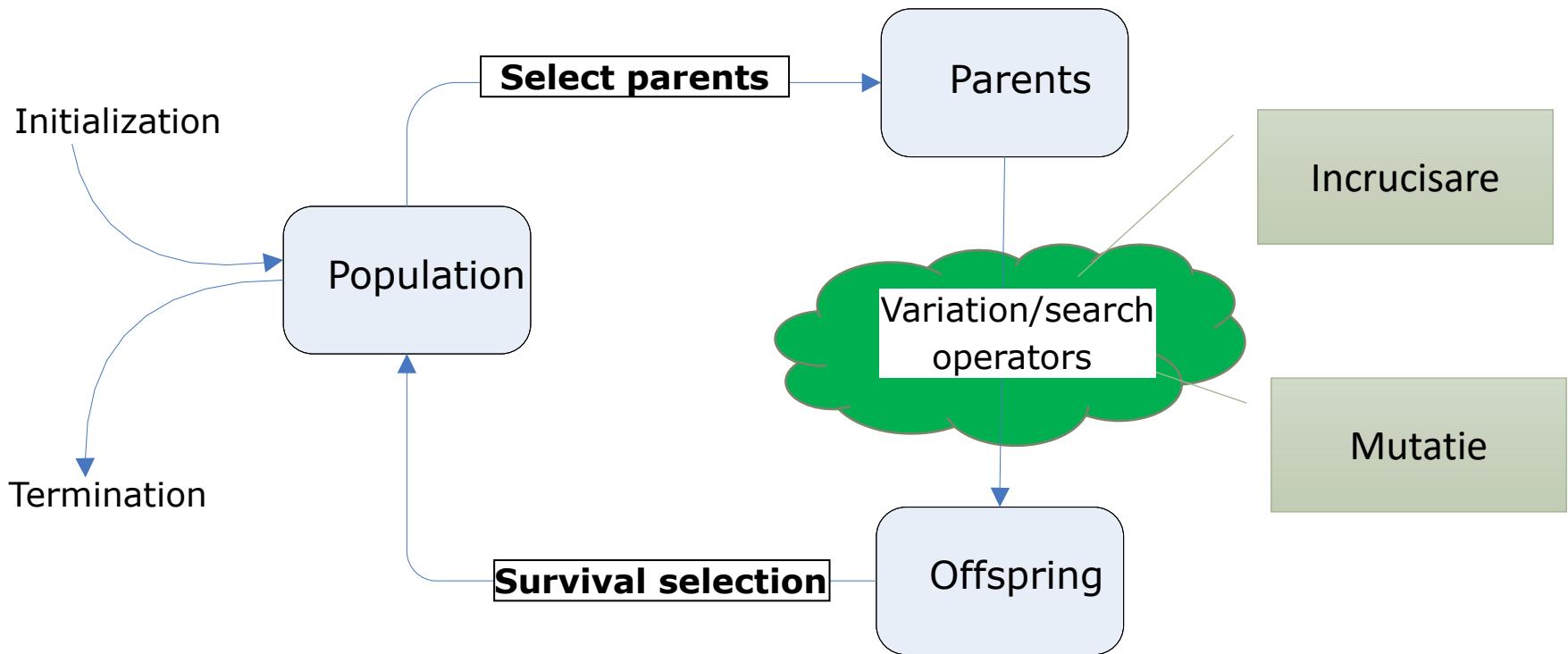
# Codificarea binara

1 | 0 | 1 | 1

Un numar intreg:  
 $1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 =$   
 $8 + 0 + 2 + 1 = 11$

Un numar real:  
 $0.5 + 11/15 = 1.23$

- $\{0,1\}^L$
- O schema este un sir format cu simbolurile 0, 1 si \*
  - Simbolul \* poate fi inlocuit cu 0 sau 1
- Exemple
  - Schema  $S = (1 * * 0)$  reprezinta 4 cromozomi
  - $X = 1 0$  este reprezentat de 4 scheme
    - $S_1 = * *; S_2 = * 0; S_3 = 1 0; S_4 = 1 *$
    - Cromozomul x este o instanta sau un reprezentant al fiecareia din schemele  $S_1 - S_4$



# Incrucisarea

- Operatorul de incrucisare (crossover) realizeaza recombinarea individelor selectate
- $r$  – lungimea cromozomilor unei populatii
- $x = x_1 x_2 \dots x_k \dots x_r$

# Incrucisarea cu un punct de taietura

- Un punct de taietura este un numar intreg  $k \in \{1, 2, \dots, r - 1\}$
- $X$  – multimea tuturor cromozomilor de lungime fixata  $r$
- $C$  – operatorul de incrusare
- $C : X \times X \rightarrow X \times X$
- $C(x, y) = (x', y')$

Parinti

$$x = x_1 x_2 \dots x_k x_{k+1} \dots x_r$$

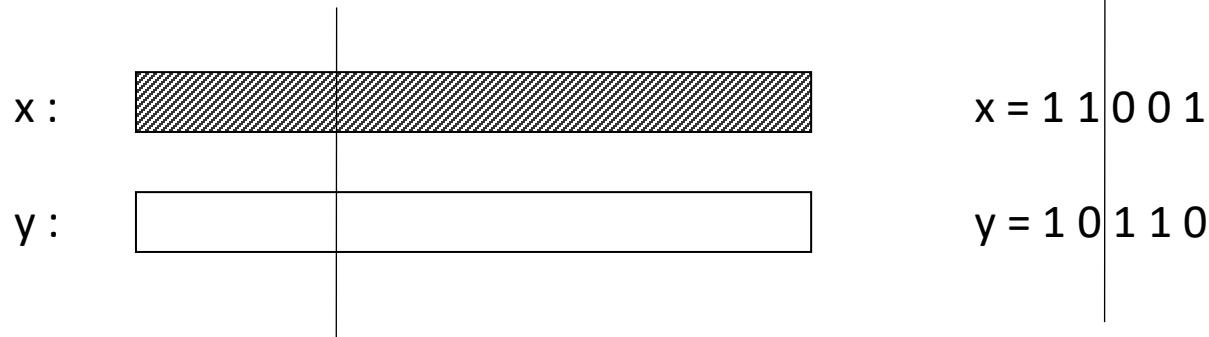
$$y = y_1 y_2 \dots y_k y_{k+1} \dots y_r$$

Cromozomii copii vor fi:

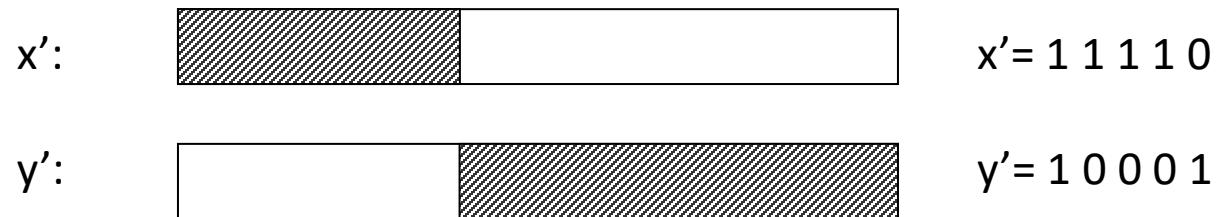
$$x' = x_1 x_2 \dots x_k y_{k+1} \dots y_r$$

$$y' = y_1 y_2 \dots y_k x_{k+1} \dots x_r$$

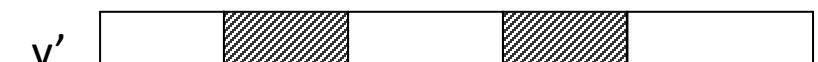
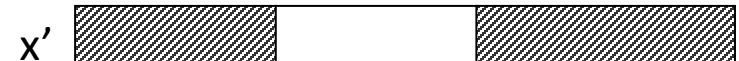
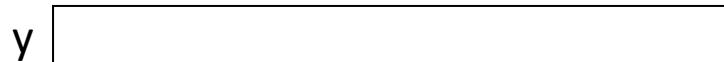
Fie cromozomii:



Dupa incruisare cu un punct de tietura k = 2 rezulta succesorii:



# Incrucisarea cu mai multe puncte de tăietura



-Anumite combinații de gene nu pot fi realizate utilizând un singur punct de tăietura

*-Exemplu:*

$$S_1 = ( 0 \ 1 \ * \ * \ * \ * \ * \ * \ 1 \ 1 )$$

$$S_2 = ( * \ * \ * \ 1 \ 0 \ 1 \ * \ * \ * \ * )$$

$$S_3 = ( 0 \ 1 \ * \ 1 \ 0 \ 1 \ * \ * \ * \ 1 \ 1 )$$

# Incrucisarea adaptiva

- Gasirea unui mecanism de a considera calitatea descendantilor obținuti prin incrusarea cu diferite puncte de taietura
- **ADAPTARE:** distributia punctelor de incrusare
  - Inregistrarea pozitiilor punctelor de taietura
  - Cele care produc descendenti cu un fitness bun vor ramane *active* (punctul va fi considerat mort altfel)

# Incrucisarea segmentata

- Numarul punctelor de tajetura variaza
- $s$  – probabilitatea ca un segment sa aiba extremitatea dreapta in oricare din pozitiile ce urmeaza inceputului sau
- Pornind de la prima pozitie  $i=1$ 
  - Se genereaza aleator  $q \in [0,1]$
  - Se genereaza aleator  $j: i < j \leq r$
  - $q =$  probabilitatea de acceptare a punctului de tajetura  $j$

## Incrucisarea cu amestec (shuffle X)

- Se amesteca aleator genele celor doi cromozomi  $x$  si  $y$  retinand pozitia initiala a fiecarei gene. Rezulta cromozomii  $x_1$  si  $y_1$ .
- Se incruciseaza cei doi cromozomi  $x_1$ ,  $y_1$  folosind un operator de incrusare (cu mai multe puncte de taietura, de exemplu). Fie  $x_2$  si  $y_2$  cei doi descendenti obisnuiti.
- Se realizeaza dezamestecarea (reordonarea) pozitiilor cromozomilor  $x_2$  si  $y_2$ . Rezulta descendantii  $x_3$  si  $y_3$ .

# Incrucisarea uniforma

- Nu foloseste puncte de taietura
- *Parametru: p* - probabilitatea ca gena unui descendente sa provina din primul sau al doilea parinte
  - $x = x_1 x_2 \dots x_k \dots x_r$
  - $y = y_1 y_2 \dots y_k \dots y_r$
  - Pentru fiecare pozitie  $i$  din  $x'$  se alege parintele care va da valoarea pozitiei respective cu prob  $p$
  - Pentru  $y'$  se ia valoarea pozitiei corespunzatoare din celalalt parinte;  
*Alternativ:* independent de  $x'$
- Poate combina caracteristici indiferent de pozitia relativa

# Rolul incrucisarii

- Prin incrucisare se pot obtine descendenți total diferiti de parintii lor
- Renuntarea la incrucisare induce, de regula, o descrestere a performantei
- Incrucisarea este responsabilă pentru accelerarea în mod semnificativ a procesului de căutare
- Mutatia (celalalt operator de căutare important) este mai degraba o metodă pentru a reintroduce diversitatea într-o populație
- Componența de căutare de mare performanță este incrucisarea
  - Combinarea rapidă a ceea ce este bun în populația initială
  - Proliferarea celor mai promitătoare blocuri constructive (conf. teorema schemelor)

# Mutatia

- Efect: schimbarea valorii unei singure pozitii (gene) dintr-un cromozom
- Operator de tip probabilist
- Probabilitatea de aplicare a operatorului se numeste *probabilitate de mutatie* si se noteaza cu  $p_m$
- Operatorul de mutatie actioneaza asupra bitilor, indiferent de pozitia lor in cromozom
- Fiecare bit al populatiei poate suferi o mutatie

# Mutatia tare

- P1. Pentru fiecare cromozom al populatiei curente si pentru fiecare pozitie a cromozomului se executa:
  - P1.1. Se genereaza un numar aleator  $q$  in intervalul  $[0,1]$ .
  - P1.2. Daca  $q < p_m$  atunci se executa mutatia pozitiei respective, schimband 0 in 1 si 1 in 0.

In caz contrar ( $q \geq p_m$ ), pozitia respectiva nu se schimba

# Mutatia slaba

- P1. Pentru fiecare cromozom al populatiei curente si pentru fiecare pozitie a cromozomului se executa:
  - P1.1. Se genereaza un numar aleator  $q$  in intervalul  $[0,1]$ .
  - P'1.2. Daca  $q < p_m$  atunci se alege aleator una din valorile 0 sau 1. Se atribuie pozitiei curente valoarea astfel selectata.  
Daca  $q \geq p_m$  atunci pozitia curenta nu se schimba.

# Mutatia neuniforma

$p_m$  depinde de generatie

- Mare in primele generatii
- Descreste cu indicele  $t$  al generatiei
- Schimbari mari in primele etape ale cautarii
- Accentul pe cautare locala in faze avansate

$$p_m(t) = p_m e^{(1-t)\beta}$$

- $\beta \geq 1$  este un parametru real
- Cu cat este mai mare – cu atat mai repede descreste probabilitatea de mutatie

# Mutatia neuniforma adaptiva

- $p_m$  depinde de *pozitia in cromozom si de generatie*
- **Exemplu:** cromozomii codifica binar numere reale
  - Daca o gena este pozitionata la inceputul unui cromozom, atunci schimbarea ei provoaca o modificare semnificativa a cromozomului
  - Elementele din spatiul starilor ce corespund celor doi cromozomi vor fi foarte diferite
  - Mutatia genelor aflate spre sfarsitul cromozomului va induce o schimbare mult mai mica
- *Cautare globala in primele etape*
- *Cautare locala in ultima parte a procesului iterativ*
- Pe masura ce indicele generatiei creste probabilitatea de mutatie a primelor gene din fiecare cromozom descreste, iar cea a ultimelor gene creste

# **Codificarea reală**

## **Incrucisare, Mutatie**

# Codificarea reală

- Utilizează numere reale pentru reprezentarea valorilor genelor
- Potrivită în special pentru rezolvarea unor probleme de optimizare în care variabilele iau valori într-un domeniu continuu ex.  $f: \mathcal{R}^n \rightarrow \mathcal{R}$

F12: Schwefel's Problem 2.13

$$F_{12}(\mathbf{x}) = \sum_{i=1}^D (\mathbf{A}_i - \mathbf{B}_i(\mathbf{x}))^2 + f\_bias_{12}, \mathbf{x} = [x_1, x_2, \dots, x_D]$$

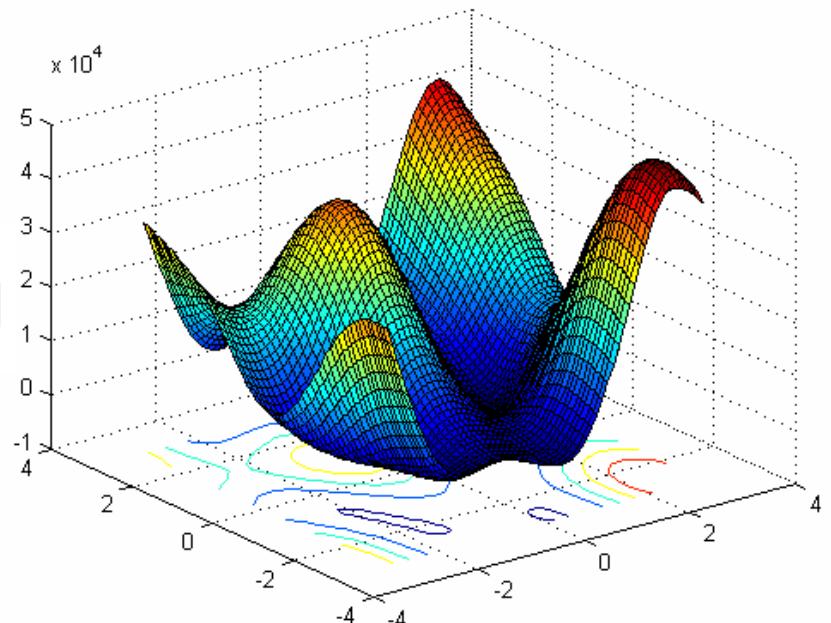
$$\mathbf{A}_i = \sum_{j=1}^D (a_{ij} \sin \alpha_j + b_{ij} \cos \alpha_j)$$

$$\mathbf{B}_i(x) = \sum_{j=1}^D (a_{ij} \sin x_j + b_{ij} \cos x_j)$$

for  $i = 1, \dots, D$

$a_{ij}, b_{ij}$  are integer random numbers in the range [-100,100]

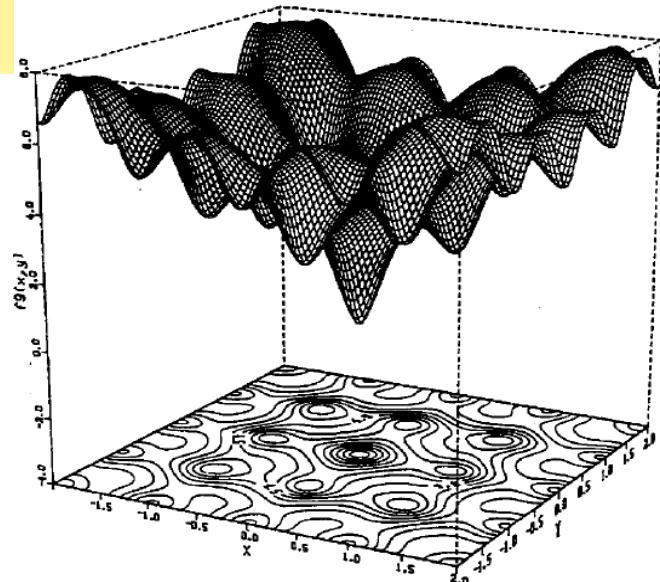
$\alpha = [\alpha_1, \alpha_2, \dots, \alpha_D], \alpha_j$  are random numbers in the range  $[-\pi, \pi]$



# Ackley's Function

$$f(\bar{x}) = -c_1 \cdot \exp \left( -c_2 \cdot \sqrt{\frac{1}{n} \sum_{i=1}^n x_i^2} \right) - \exp \left( \frac{1}{n} \cdot \sum_{i=1}^n \cos(c_3 \cdot x_i) \right) + c_1 + 1$$

$$c_1 = 20, c_2 = 0.2, c_3 = 2\pi$$



## Reprezentare

Un cromozom este un vector cu componente reale.

$$x = \{x_1, x_2, \dots, x_n\}$$

Pozitia  $i$  reprezinta valoarea genei  $i$  in cromozomul  $x$ .

$$x_i \in R$$

$$x \in R^n$$

# Incrucisare pentru codificarea reală

- Incrusarea cu mai multe puncte de tăietura este ușor aplicabilă dar neadecvată

=> *Redefinirea operatorului de incrusare*

- Incrusarea discreta
- Incrusarea continuă
- Incrusarea convexă
- Operatorul SBX

# Incrucisarea discreta

- ANALOG cu *incrucisarea uniformă*:  $\mathbf{x}_i \text{ or } \mathbf{y}_i$
- Pentru fiecare pozitie  $i$  a primului descendente se alege (cu o probabilitate fixată  $p$ ) parintele a carui genă (din pozitia  $i$ ) va fi transmisa acestui descendente
- $p=0.5$  - rolul celor doi parinti este simetric

## Exemplu

Parinti

$$\mathbf{x} = (x_1, x_2, x_3, x_4, x_5)$$

$$\mathbf{y} = (y_1, y_2, y_3, y_4, y_5).$$

Incrucisare discreta cu  $p=0.5$ :

$$x_1 = (x_1, x_2, y_3, x_4, y_5)$$

$$y_1 = (y_1, y_2, x_3, y_4, x_5)$$

# Incrucisarea continua (sau medie)

- Se aleg aleator (cu o probabilitate fixata p) anumite pozitii
- Genele corespunzatoare in descendenți vor fi media aritmetica a genelor corespunzatoare ale parintilor

## *Exemplu*

Parinti

$$\begin{aligned}x &= (x_1, x_2, x_3, x_4, x_5) \\y &= (y_1, y_2, y_3, y_4, y_5).\end{aligned}$$

Incrucisare medie pt pozitiile 3 si 5:

$$\begin{aligned}x_1 &= (x_1, x_2, (x_3 + y_3)/2, x_4, (x_5 + y_5)/2) \\y_1 &= (y_1, y_2, (x_3 + y_3)/2, y_4, (x_5 + y_5)/2)\end{aligned}$$

# Incrucisarea medie completa

- Produce un singur descendant ale carui gene reprezinta media aritmetica a valorilor genelor corespunzatoare din cei doi parinti
- Fiecare gena  $i$  a descendantului va fi:

$$z_i = \frac{1}{2}(x_i + y_i)$$

# Incrucisarea convexa (sau intermediara sau aritmetica)

- Parintii nu au aceeasi pondere in obtinerea descendantului
- Combinatie convexa a genelor parintilor
- Gena de la pozitia  $i$  a unicului descendant:

$$z_i = \alpha \cdot x_i + (1 - \alpha) \cdot y_i,$$

$$\alpha : 0 \leq \alpha \leq 1$$

- $\alpha$  constant: *incrucisare convexa uniforma*
- $\alpha$  depinde de generatie: *incrucisare convexa neuniforma*
- $\alpha$  generat aleator pentru fiecare pereche de parinti
- $\alpha$  isi schimba valoarea pentru fiecare gena

$$u_i = \alpha \cdot x_i + (1 - \alpha) \cdot y_i,$$

$$v_i = \alpha \cdot y_i + (1 - \alpha) \cdot x_i.$$

# Incrucisarea convexa unica

- Parinti:  $x = \langle x_1, \dots, x_n \rangle$ ;  $y = \langle y_1, \dots, y_n \rangle$
- Se alege aleator o gena - pozitia ( $k$ )

$$\langle x_1, \dots, x_k, \alpha \cdot y_k + (1 - \alpha) \cdot x_k, \dots, x_n \rangle$$

Exemplu pentru  $\alpha = 0.5$

0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
-----	-----	-----	-----	-----	-----	-----	-----	-----

0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.5	0.9
-----	-----	-----	-----	-----	-----	-----	-----	-----



0.3	0.2	0.3	0.2	0.3	0.2	0.3	0.2	0.3
-----	-----	-----	-----	-----	-----	-----	-----	-----

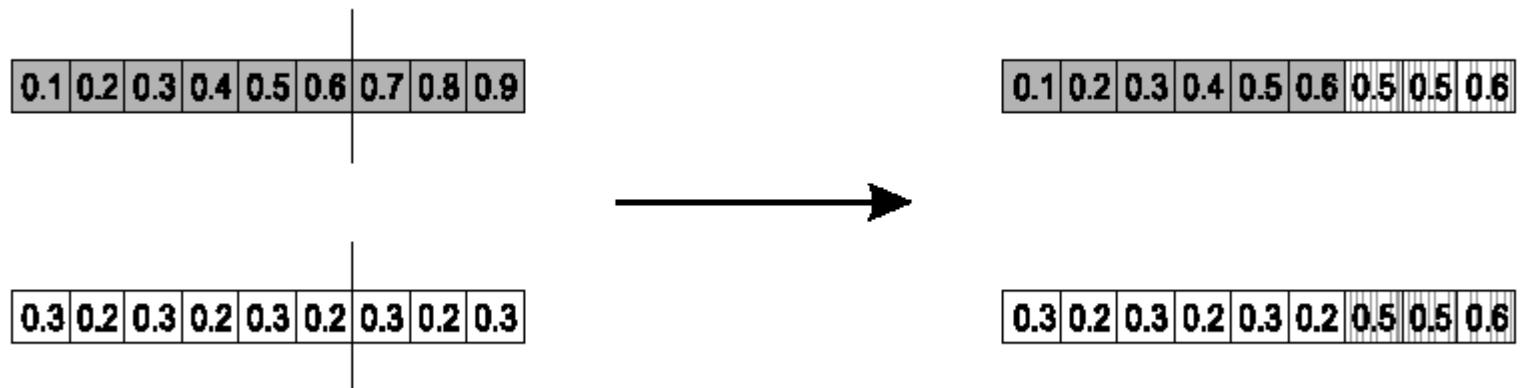
0.3	0.2	0.3	0.2	0.3	0.2	0.3	0.5	0.3
-----	-----	-----	-----	-----	-----	-----	-----	-----

# Incrucisarea convexa simplă

- Parinti:  $x = \langle x_1, \dots, x_n \rangle$ ;  $y = \langle y_1, \dots, y_n \rangle$
- Se alege aleator o genă – pozitia ( $k$ ) – și după aceasta pozitie se combina genele

$$\left\langle x_1, \dots, x_k, \alpha \cdot y_{k+1} + (1 - \alpha) \cdot x_{k+1}, \dots, \alpha \cdot y_n + (1 - \alpha) \cdot x_n \right\rangle$$

Exemplu pentru  $\alpha = 0.5$



# SBX (Simulated Binary Crossover)

- Simuleaza incrusisarea cu un punct de tajetura (codificarea binara)
- Utilizarea unei distributii de probabilitate continue ( $\beta$  – un numar real pozitiv):

$$P(\alpha, \beta) = \begin{cases} \frac{\alpha+1}{2} \cdot \beta^n, & 0 \leq \beta \leq 1 \\ \frac{\alpha+1}{2} \cdot \frac{1}{\beta^{n+2}}, & \beta > 1 \end{cases}$$

Parinti foarte diferiti – *solutii noi departate de parinti*  
Parinti apropiati – *descendenti apropiati de parinti*

## SBX pentru $x_1$ si $x_2$ (parinti)

P1. Se genereaza un numar aleator  $u \in [0,1]$

P2. Se determina un parametru  $\delta$  astfel incat

$$\int_0^\delta P(\alpha, \beta) d\beta = u$$

P3. Valoarea calculata  $\delta$  se foloseste pentru obtinerea  
descendentilor  $y_1$  si  $y_2$  :

$$y_1 = \frac{1}{2} \left( x_1 + x_2 - \delta |x_1 - x_2| \right),$$

$$y_2 = \frac{1}{2} \left( x_1 + x_2 + \delta |x_1 - x_2| \right).$$

# Mutatia in codificarea reala

- Un cromozom codifica n parametri (cromozomul contine n gene)
- Fiecare pozitie  $i$  (- parametru) – numar real cu valori intr-un domeniu  $[Ll_i, Ls_i]$
- **Mutatia uniforma**
- **Mutatia neuniforma**

# Mutatia uniforma

- Inlocuieste o singura gena a cromozomului cu un numar real generat aleator
- Parinte
  - $x = (x_1, x_2, \dots, x_n)$
- Dupa mutatie pentru pozitia  $i$ 
  - $x' = (x_1, x_2, \dots, x_i', \dots, x_n)$ ,  $1 \leq i \leq n$ ,
  - $x_i' \in [L_{I_i}, L_{S_i}]$ , uniform aleator selectat
- Var: gena  $i$  determinata aleator
- Var: fiecare pozitie sufera mutatie cu probabilitatea  $p_m$

# Mutatia neuniforma

- Genele sufera modificari importante in primele generatii



- La fiecare generatie  $t$  se genereaza aleator doi parametri:
  - $p$  - indica natura schimbarii
    - $p=1$  indica o crestere a valorii unei gene
    - $p=-1$  indica o descrestere
  - $r$  - determina amplitudinea schimbarii
    - $r$  este un numar aleator in intervalul  $[0,1]$ , urmand o distributie uniforma

$$\mathbf{x} = (x_1, x_2, \dots, x_n)$$

$x_i$  gena aleasa pentru mutatie

$$x_i' = x_i + (x_{\max} - x_i) \left[ 1 - r^{\left(1 - \frac{t}{T}\right)} \right], p = 1$$

$$x_i' = x_i - (x_i - x_{\min}) \left[ 1 - r^{\left(1 - \frac{t}{T}\right)} \right], p = -1$$

unde

$x_{\min}$  si  $x_{\max}$  semnifica marginea inferioara si respectiv superioara a variabilei (parametrului)  $x_i$

T este indicele generatiei pentru care amplitudinea mutatiei se anuleaza (generatiile ulterioare nu vor mai suferi mutatii) - T poate fi numarul maxim de generatii

$$\mathbf{x}' = (x_1, x_2, \dots, x_i', \dots, x_n)$$

# **Codificarea specifică: permutari**

**Incrucisare, Mutatie**

# Codificarea specifică: permutari

- Reprezentarea reflectă specificul problemei
- Problema de planificare a sarcinilor: *o lista de  $j$  sarcini trebuie ordonată pentru a fi executate într-o fabrică*
  - Ordinea pozitiei - importanța
- TSP
  - Pozitiile *adiacente* - importante
- Reprezentare: **[1,2,...,j]**
  - Ordinea din permutare corespunde ordinii aplicării sarcinilor
  - Este posibilă existența a mai multor instante pentru anumite sarcini

# Incrucisare pentru permutari

- Operatorii de incrusare clasici conduc de cele mai multe ori la solutii inadmisibile



- Incrucisarea bazata pe ordonare (Order Crossover)
- Incrucisarea PMX (Partially Mapped Crossover)
- Incrucisarea ciclu (Cycle Crossover)

# OX (Order Crossover)

- Alege o subruta dintr-un parinte si pastreaza ordinea relativa a oraselor din celalalt parinte
- *Algoritm:*
  1. Alege aleator o parte (i...j) din primul parinte
  2. Pentru primul descendent
    - 2.1 Copiaza partea (i...j)
    - 2.2 Seteaza celelalte pozitii astfel:
      - Incepand de la pozitia imediat urmatoare lui j
      - Folosind **ordinea** din al doilea parinte
      - Continuand circular pana la pozitia dinaintea lui i
  3. Al doilea descendent se creaza similar cu primul dar cu rolurile parintilor schimbat

# OX

$$p_1 = (1 \ 2 \ 3 \mid 4 \ 5 \ 6 \ 7 \mid 8 \ 9)$$

$$p_2 = (4 \ 5 \ 2 \mid 1 \ 8 \ 7 \ 6 \mid 9 \ 3)$$

Segmentele dintre cele 2 puncte de tăietura sunt copiate

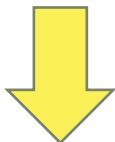
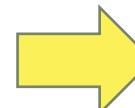
$$o_2 = (3 \ 4 \ 5 \mid 1 \ 8 \ 7 \ 6 \mid 9 \ 2)$$



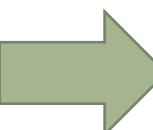
$$o_1 = (x \ x \ x \mid 4 \ 5 \ 6 \ 7 \mid x \ x)$$

$$o_2 = (x \ x \ x \mid 1 \ 8 \ 7 \ 6 \mid x \ x)$$

**Ordine din  $p_1$ :**  
8-9-1-2-3-4-5-6-7  
**Minus orase existente:**  
9-2-3-4-5



**Ordine din  $p_2$ :**  
9-3-4-5-2-1-8-7-6  
**Minus orase existente:**  
9-3-2-1-8



$$o_1 = (2 \ 1 \ 8 \mid 4 \ 5 \ 6 \ 7 \mid 9 \ 3)$$

Orasele 9-3-2-1-8 copiate incepand de la al doilea punct de tăietura

# PMX (Partially Mapped Crossover)

Algoritm PMX pentru P1 si P2 (parinti)

1. Alege aleator un segment din P1
2. Incepand de la primul punct de taietura cauta elementele in acel segment din P2 care nu au fost copiate
3. Pentru fiecare astfel de element  $i$  verifica in descendant ce element  $j$  a fost copiat in locul lui din P1
4. Plaseaza  $i$  in pozitia ocupata  $j$  in P2 (j este deja in offspring)
5. Daca locul ocupat de  $j$  in P2 a fost deja ocupat in descendant  $k$ , pune  $i$  in pozitia ocupata de  $k$  in P2
6. Restul pozitiilor din descendant sunt copiate din P2.

*Al doilea descendant este creat analog.*

# PMX

- Un offspring se creaza prin alegerea unui subruta de la un parinte si pastrarea ordinii si pozitiei cat mai multor orase din celalalt parinte
- Subruta se creeaza selectand aleator 2 puncte de taietura

$$p_1 = (1 \ 2 \ 3 \ | \ 4 \ 5 \ 6 \ 7 \ | \ 8 \ 9)$$

$$p_2 = (4 \ 5 \ 2 \ | \ 1 \ 8 \ 7 \ 6 \ | \ 9 \ 3)$$



$$o_1 = (x \ x \ x \ | \ 1 \ 8 \ 7 \ 6 \ | \ x \ x)$$

$$o_2 = (x \ x \ x \ | \ 4 \ 5 \ 6 \ 7 \ | \ x \ x)$$

$$o_1 = (4 \ 2 \ 3 \ | \ 1 \ 8 \ 7 \ 6 \ | \ 5 \ 9)$$

$$o_2 = (1 \ 8 \ 2 \ | \ 4 \ 5 \ 6 \ 7 \ | \ 9 \ 3)$$

*$o_1$ :*  
- primul x trebuie sa fie 1 => se mapeaza la 4  
- al doilea x trebuie sa fie 8 => se mapeaza la 5



## Mappings

- |   |   |   |
|---|---|---|
| 1 | ↔ | 4 |
| 8 | ↔ | 5 |
| 7 | ↔ | 6 |
| 6 | ↔ | 7 |

$$o_1 = (x \ 2 \ 3 \ | \ 1 \ 8 \ 7 \ 6 \ | \ x \ 9)$$

$$o_2 = (x \ x \ 2 \ | \ 4 \ 5 \ 6 \ 7 \ | \ 9 \ 3)$$

*Completam cu orase din parintii originali daca nu exista conflicte*

# CX (Cycle Crossover)

- Un offspring se creeaza astfel incat fiecare oras si pozitia lui vine de la unul din parinti
- **Algoritm:**
  1. Construirea unui ciclu de alele din P1 astfel:
    - 1.1 Prima pozitie – prima poz din P1.
    - 1.2 Merg *la aceeasi pozitie* din P2.
    - 1.3 Merg la pozitia *cu aceeasi valoare* in P1.
    - 1.4 Adaug aceasta pozitie in ciclu.
    - 1.5 Repeta pasii 1.1-1.4 pana cand ajung din nou la prima pozitie din P1.
  2. Copiaza valorile din pozitiile din *primul* ciclu folosind primul parinte .
  3. Urmatorul ciclu din al doilea parinte

# CX

Luam primul oras  
din primul parinte

$$p_1 = (1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9)$$

$$p_2 = (4 \ 1 \ 2 \ 8 \ 7 \ 6 \ 9 \ 3 \ 5)$$



$$o_1 = (1 \ x \ x \ x \ x \ x \ x \ x \ x)$$

1 -> 4 ...

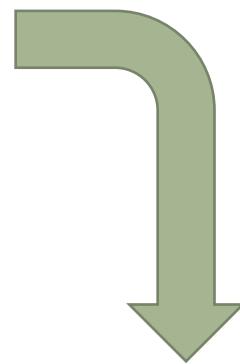
$$o_1 = (1 \ x \ x \ 4 \ x \ x \ x \ x \ x)$$

4 -> 8 ...

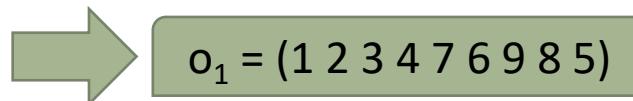
$$o_1 = (1 \ x \ x \ 4 \ x \ x \ x \ 8 \ x)$$

8 -> 3 -> 2 ...

$$o_1 = (1 \ 2 \ 3 \ 4 \ x \ x \ x \ 8 \ x)$$



$$o_2 = (4 \ 1 \ 2 \ 8 \ 5 \ 6 \ 7 \ 3 \ 9)$$



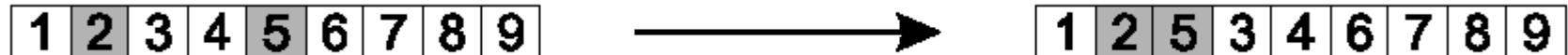
Restul oraselor in  
ordinea din  $p_2$

# Mutatie pentru permutari

- Mutatia standard duce la solutii invalide
  - [1,2,3,4,5]
  - mutatie pentru pozitia 2: [1,5,3,4,5] !!
  - Trebuie alterate valorile in cel putin 2 pozitii
- Probabilitatea de mutatie se refera acum la probabilitatea de a muta un cromozom nu o pozitie din cromozom
- Insert mutation
- Swap mutation
- Inversion mutation
- Scramble mutation

# Mutatia inserare (insert)

- Selecteaza aleator 2 pozitii
- Muta valoarea din a doua pozitie in pozitia imediat urmatoare primei pozitii selectate
- Translateaza toate celelalte pozitii pentru a pastra o permutare valida



# Mutatia interschimare (swap)

- Selecteaza aleator 2 pozitii
- Interschimba valorile
- Ordinea afectata mai tare

1	2	3	4	5	6	7	8	9
---	---	---	---	---	---	---	---	---



1	5	3	4	2	6	7	8	9
---	---	---	---	---	---	---	---	---

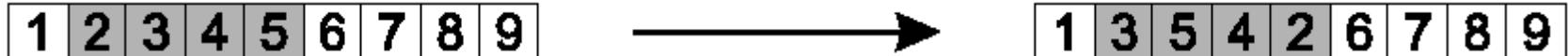
# Mutatia inversiune (inversion)

- Selecteaza aleator 2 pozitii
- Inverseaza ordinea pozitiilor in segmentul astfel determinat
- Ordinea afectata mai tare, elementele adiacente pastrate in mare parte



# Mutatia amestec (scramble)

- Selecteaza aleator un segment din permutare
- Reordoneaza aleator pozitiile din segment



# Remarci AE

- *AE sunt conceptual simpli:* genereaza o **populatie** de solutii potentiiale pentru problema data, aplica niste operatori de **variatie** pentru a crea solutii noi din cele vechi, si aplica un mecanism de **selectie** pentru a pastra cele mai bune solutii gasite
  - Pentru orice problema, un AE trebuie proiectat tinand cont de:
    - Obiectivul problemei si evaluarea unei solutii
    - Reprezentarea solutiilor problemei
  - Daca exista cunostinte despre zona unde se poate gasi o solutie optima, este bine sa fie incorporate in AE (initializare, variatie)
- ✓ Toate elementele AE – *reprezentare, initializare, evaluare, variatie, selectie* – trebuie considerate impreuna, nu separat!

# Proiectarea AE

- Reprezentare
- Evaluarea indivizilor: functia de fitness
- Specificarea unor operatori potriviti pentru
  - Mutatie
  - Incrucisare
- Selectia:
  - Selectia indivizilor ce vor fi parinti
  - Selectia indivizilor pentru supravietuire
- Start: Initializare
- Stop: Criteriul de terminare
- Pasii algoritmului: ce se intampla intr-o generatie?

# Incrucisarea

## • Codificarea binara

- Incrucisarea cu un punct de taietura / mai multe puncte de taietura
- Incrucisarea adaptiva (**adaptare**: distributia punctelor de incrucisare)
- Incrucisarea segmentata (numarul pct de taietura variaza)
- Incrucisarea cu amestec
- Incrucisarea uniforma

## • Codificarea reala

- Incrucisarea discreta
- Incrucisarea continua (sau medie) si medie completa
- Incrucisarea convexa (sau intermediara sau aritmetica)
- Operatorul SBX

## • Codificarea prin permutari

- Incrucisarea bazata pe ordonare (**Order Crossover**)
- Incrucisarea PMX (**Partially Mapped Crossover**)
- Incrucisarea ciclu (**Cycle Crossover**)

# Mutatia

- **Codificarea binara**

- Mutatia tare
- Mutatia slaba
- Mutatia neuniforma ( $p_m$  depinde de *generatie*)
- Mutatia neuniforma adaptiva ( $p_m$  depinde si de *pozitia in cromozom*)

- **Codificarea reala**

- Mutatia uniforma
- Mutatia neuniforma

- **Codificarea prin permutari**

- **Mutatia inserare (Insert mutation)**
- **Mutatia interschimbare (Swap mutation)**
- **Mutatia inversiune (Inversion mutation)**
- **Mutatia amestec (Scramble mutation)**

# Selectia

- Selectia determinista
  - Selectia  $(\mu+\lambda)$
  - Selectia  $(\mu,\lambda)$
- **Selectia stocastica**
  - Selectia proporcionala
  - Selectia prin ordonare
  - Selectia turnir
- **Alte strategii de selectie**
  - Elitism
  - Folosirea unei rate de inlocuire
  - Inlocuirea asincrona a indivizilor (vs generationala)

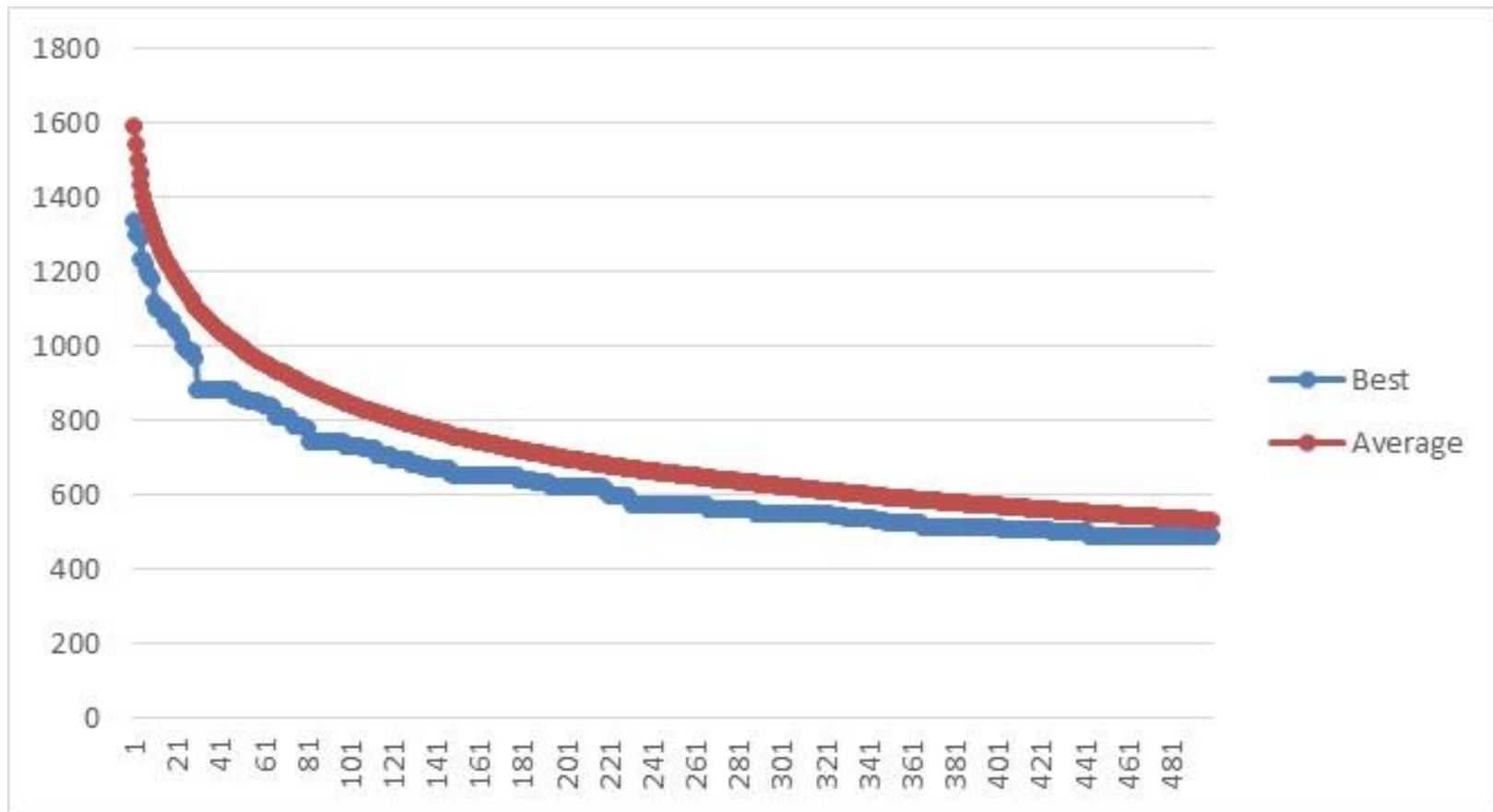
# Experimente

- Nici o concluzie nu poate fi trasa dintr-o singura rulare
  - Numar suficient de rulari independente
  - Masuri statistice: media, deviatia standard
  - Teste statistice
- Comparatii cu alti algoritmi
- Ce masuram?
  - Rezultatul mediu obtinut intr-un anumit timp
  - Timpul mediu necesar obtinerii unui anumit rezultat
  - Proportia de rulari in care s-a obtinut o anumita solutie
  - Cea mai buna solutie din n rulari
- Time units?
  - **Timp necesar:** Depinde de calculator, de implementare,...
  - **Numar generatii:** Daca alti parametri se schimba ex. Marimea populatiei...?

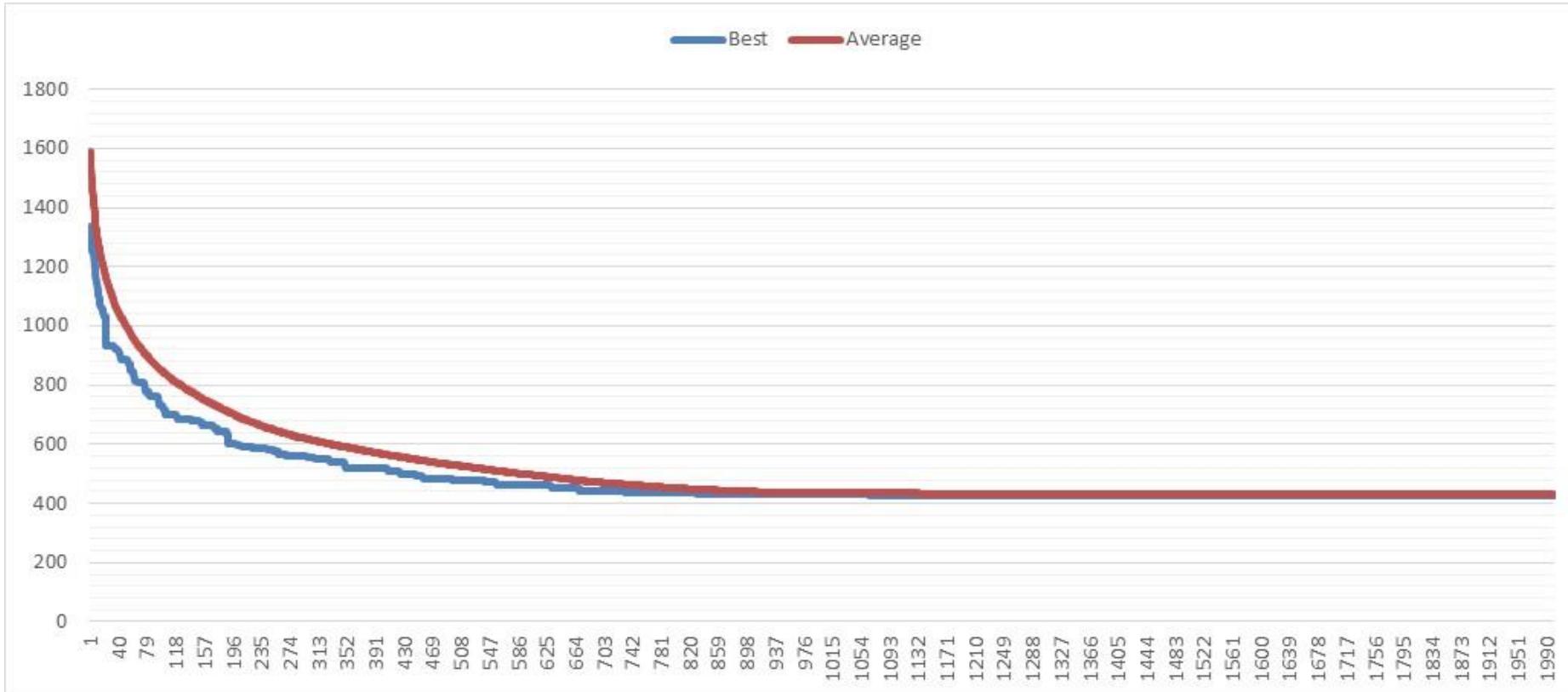
# Masurarea performantei

- Numarul mediu de evaluari pana la solutie (MES)
- Rata de succes (RS)
  - % de rulari in care o solutie acceptabila este gasita
- Media fitness-ului cel mai bun (MBF – mean best fitness)
  - Cel mai bun fitness dintr-o rulare, medie peste toate rularile

# Population 1000, Generations 500



# Population 1000, Generations 2000





**BABEŞ-BOLYAI UNIVERSITY**  
Faculty of Mathematics and Computer Science



# Inteligentă Artificială

7: *Swarm Intelligence*

Camelia Chira

[cchira@cs.ubbcluj.ro](mailto:cchira@cs.ubbcluj.ro)

# Algoritmi inspirati de natura

- Algoritmii evolutivi simulează evoluția
- Algoritmii inspirați de comportamentul de grup simulează adaptarea colectivă și procesele sociale dintr-un colectiv
  - **Particle Swarm Optimisation (PSO)**  
<http://www.youtube.com/watch?feature=endscreen&v=JhZKc1Mgub8&NR=1>  
<https://www.youtube.com/watch?v=TWqx57CR69c>
  - **Ant Colony Optimisation (ACO)**  
[http://www.youtube.com/watch?v=jrW\\_TTxP1ow](http://www.youtube.com/watch?v=jrW_TTxP1ow)



# Bibliografie

- Eric Bonabeau, Marco Dorigo, Guy Theraulaz, *Swarm Intelligence: From Natural to Artificial Systems*, Santa Fe Institute, Studies in the Science of Complexity, Oxford University Press, 1999.
- Camazine S., Deneubourg J.-L., Franks N. R., Sneyd J., Theraulaz G., Bonabeau E., *Self-Organization in Biological Systems*. Princeton Studies in Complexity, Princeton University Press, 2001
- James Kennedy, Russel Eberhart, *Particle Swarm Optimisation*, Proceedings of IEEE International Conference on Neural Networks. IV. pp. 1942–1948, 1995.
- Marco Dorigo, Christian Blum, *Ant colony optimization theory: A survey*, Theoretical Computer Science 344 (2005) 243 – 27.

# Swarm Intelligence

## Inteligenta de grup

- Fenomene colective naturale care implica interconexiuni stranse intre indivizi
- **Swarm intelligence (SI)** describes the collective behavior of decentralized, self-organized systems, natural or artificial
- *“any attempt to design algorithms or distributed problem-solving devices inspired by the collective behavior of social insect colonies and other animal societies”* [Bonabeau, Dorigo, Theraulaz, 1999]
- Self-organization
- Stigmergy

# Sursa de inspiratie

- Comportament biologic real
  - Insecte sociale – furnici, termite, albine, viespi
  - Formarea de **roi, turma, stol**
- **Swarming (~gruparea)**
  - Agregarea animalelor similare – in general a celor care au aceeasi directie
    - Termitele *swarm* pentru a construi colonii
    - Pasarile *swarm* pentru a gasi hrana
    - Albinele *swarm* in scopul reproducerii
  - De ce?
    - Pentru a cauta mai bine
    - Pentru a migra
    - Ca si mecanism de aparare impotriva pradatorilor

# Caracteristici comune - swarming

- Reguli simple pentru fiecare individ
- Nu există control centralizat
  - Controlul este complet distribuit între membrii unui grup
  - Decentralizat, deci robust
- Comunicarea între indivizi
  - La nivel local
  - Indirectă (stigmergy)
- Comportament global
  - Emergent
  - Transcede comportamentul individual
  - Se adaptează ușor schimbarilor din mediu

# Swarm Intelligence (SI): Scurt Istoric

- Beni and Wang (1989):
  - Au introdus termenul in contextul **automatelor celulare**
  - Control decentralizat, auto-organizare, simplitate la nivel individual
- Bonabeau, Dorigo si Theraulaz (1999)
  - *“any attempt to design algorithms or distributed problem-solving devices inspired by the collective behavior of social insect colonies and other animal societies”*
- Beni (2004)
  - Intelligent swarm = a group of non-intelligent robots

# Algoritmi inspirati de natura

- Grup (roi - Swarm)
  - O colecție aparent dezorganizată de indivizi care se mișcă tinzând să se grupeze, dar fiecare individ pare să se miște într-o direcție oarecare
  - În interiorul colecției apar anumite procese sociale
  - Colecția este capabilă să efectueze sarcini complexe
    - fără nici o ghidare sau control extern
    - fără nici o coordonare centrală
  - Colecția poate atinge performanțe care nu pot fi atinse de indivizi în izolare
- Adaptare colectivă - auto-organizare
- Stigmergy

# Auto-organizare

- *Self-organization consists of set of dynamical mechanisms whereby structure appears at the global level as the result of interactions among lower-level components. The rules specifying the interactions among the system's constituent units are executed on the basis of purely local information, without reference to the global pattern, which is an emergent property of the system rather than a property imposed upon the system by an external ordering influence [Bonabeau et al., 1997]*
- Un mecanism dinamic prin care o structura globala emerge din interactiuni locale
- Interactiuni multiple
- Caracter aleator
- Feedback pozitiv / negativ

# Stigmergy

- ▶ P.P. Grassé, 1959

La coordination des taches, la regulation des constructions ne dépendent pas directement des ouvriers, mais des constructions elles-mêmes. *L'ouvrier ne dirige pas son travail, il est guidé par lui. C'est à cette stimulation d'un type particulier que nous donnons le nom du STIGMERGIE*

Derivat din cuvintele grecesti *stigma* (mark, sign) si *ergon* (work, action)

- Comunicare indirectă - doi indivizi comunică astfel:
  - Un individ modifică mediul la un moment dat
  - Celalalt individ răspunde la mediul modificat la un moment de timp ulterior

# Algoritmi SI

- **Particle Swarm Optimization (PSO)**

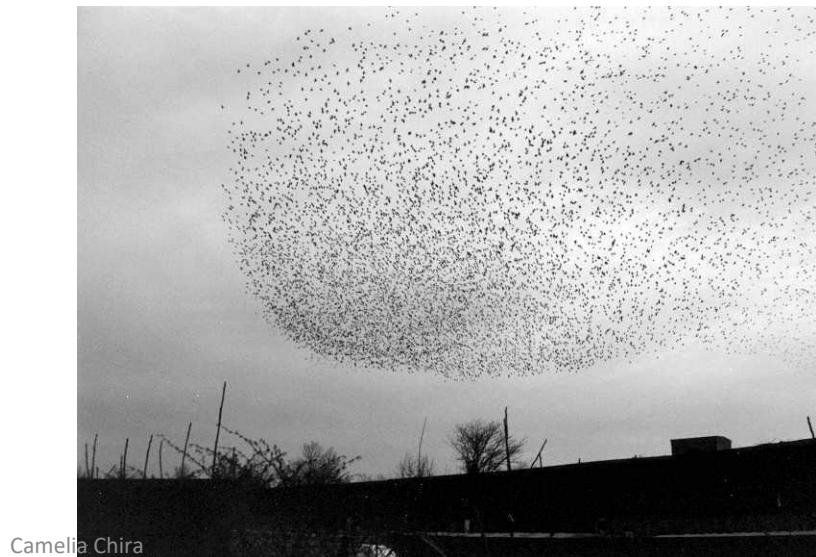
- Bio: Formarea de grupuri în stoli, turme
- Comunicare tip broadcast

- **Ant Colony Optimization (ACO)**

- Bio: Comportamentul real al coloniilor de furnici
- Stigmergy

# Particle Swarm Optimization (PSO)

- Algoritm propus de Kennedy si Eberhart in 1995
  - <http://www.particleswarm.info/>
- Publicatii
  - Kennedy, J.; Eberhart, R.C. (2001). *Swarm Intelligence*. Morgan Kaufmann.
  - Poli, R. (2008). ["Analysis of the publications on the applications of particle swarm optimisation"](#). *Journal of Artificial Evolution and Applications*: 1-10
- Probleme de optimizare
- Exemplu PSO
  - Gruparea pasarilor in stoluri

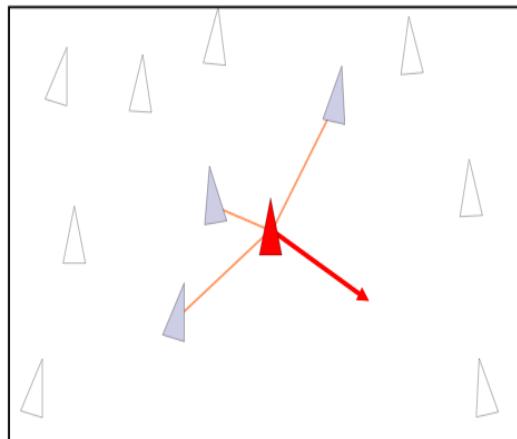


# PSO - exemplu in natura: bird flocking

- Modelul ‘Boids’ (=bird-oids=bird-like) propus de Reynolds

**Regula 1:**

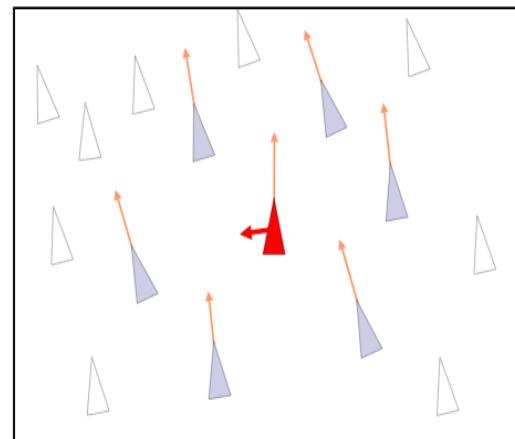
EVITAREA  
COLIZIUNILOR



...cu pasarile din  
vecinatate

**Regula 2:**

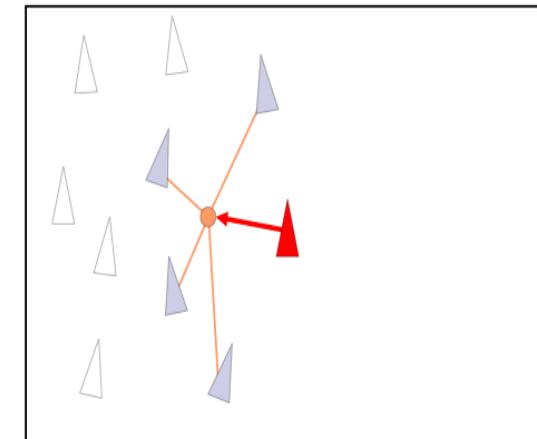
AJUSTAREA VITEZEI SI  
DIRECTIEI



...dupa cea a  
vecinilor

**Regula 3:**

ATRACTIA SPRE  
CENTRUL GRUPULUI



...prin atractia spre  
pasarile vecine

# Termeni PSO

- Populatie
  - Constituita din  $m$  particule care cauta solutia optima
  - Cooperare
- Particula
  - Corespunde unei posibile solutii
  - Se misca si are o viteza
  - Retine **locul (pozitia)** unde a obtinut cele mai bune rezultate
  - Are asociata o vecinatate de particule
- Particulele coopereaza
  - Schimbă informații (legate de descoperirile făcute în locurile deja vizitate) între ele
  - Fiecare particulă știe fitnessul vecinilor ei a.î. poate folosi poziția celui mai bun vecin pentru a-și ajusta propria viteză

# Tehnica PSO

- Fiecare particula
  - Evalueaza functia de optimizat in fiecare punct din spatiu in care ajunge
  - Retine cea mai buna valoare gasita- **pbest**
- Cea mai buna pozitie globala gasita de unul din membrii grupului – **gbest**
  - Informatie accesibila tuturor particulelor
- Cautare **cooperativa** – ghidata de calitatea relative a indivizilor

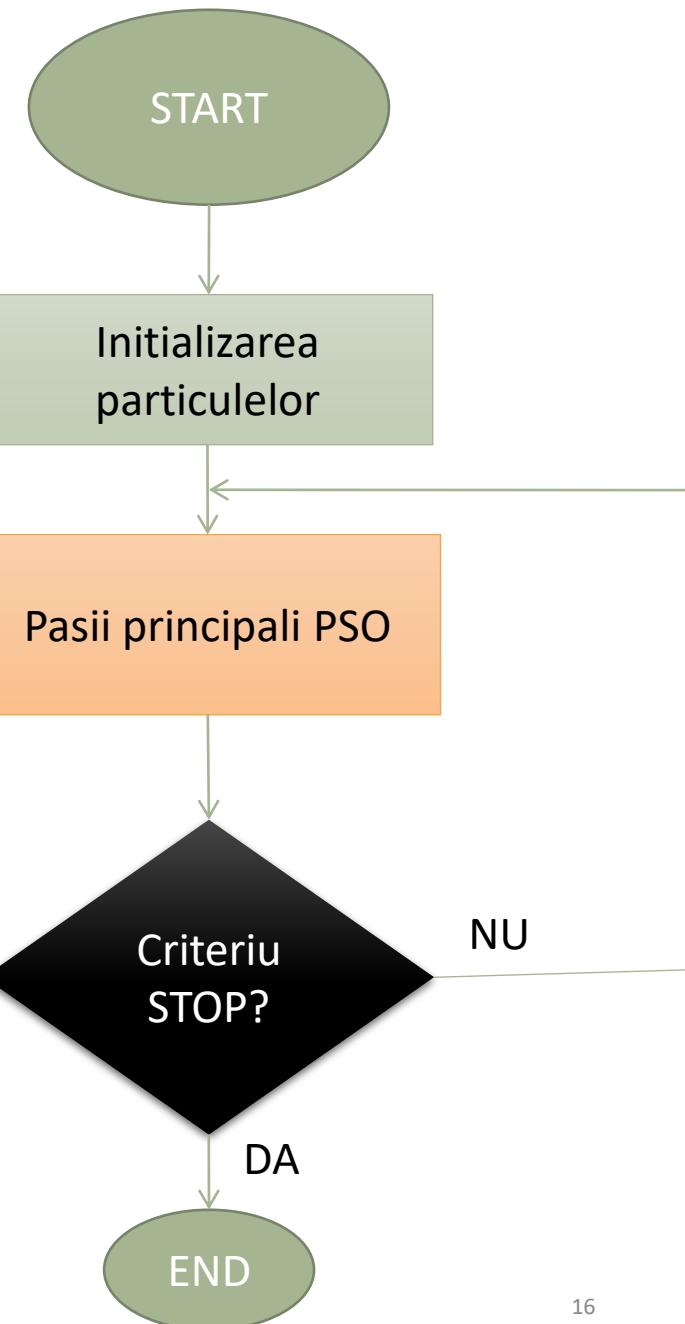
# PSO

## Crearea populației inițiale de particule

- Poziții aleatoare
- Viteze nule/aleatoare

- Evaluarea particulelor
- Pentru fiecare particulă :
  - Actualizarea memoriei
    - Stabilirea celei mai bune particule din swarm **gbest** / dintre particulele vecine **ibest**
    - Stabilirea celei mai bune poziții (cu cel mai bun fitness) în care a ajuns până atunci – **pbest**
  - Modificarea vitezei
  - Modificarea poziției

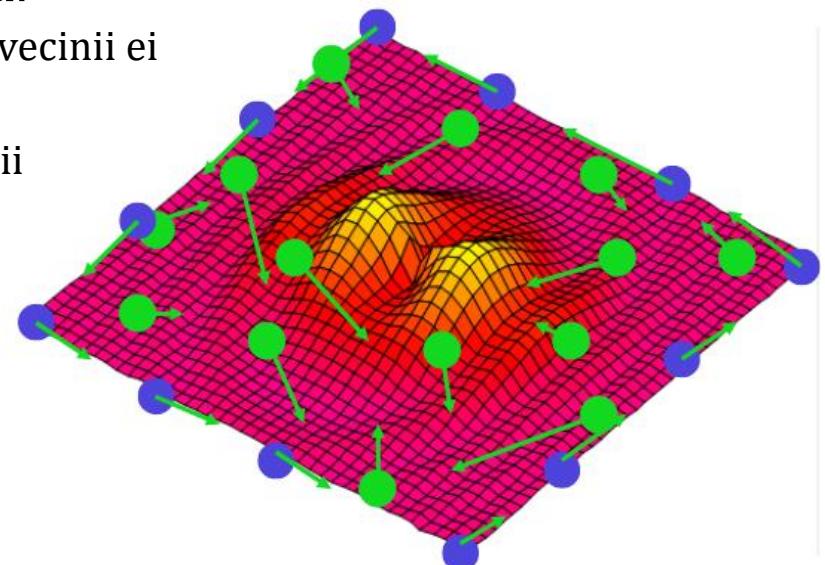
Dacă nu se îndeplinesc condițiile de oprire,  
se revine la pasul 2, altfel STOP



# Algoritm PSO

## 1. Crearea populației inițiale de particule

- Fiecare particulă are asociată
  - o poziție – potențială soluție a problemei
  - o viteză – modifică o poziție în altă poziție
  - o funcție de calitate (fitness)
- Fiecare particulă trebuie să poată:
  - interacționa (schimba informații) cu vecinii ei
  - memora o poziție precedentă
  - utilizeaza informațiile pentru a lua decizii
- Inițializarea particulelor
  - poziții aleatoare
  - viteze nule/aleatoare



# **Algoritm PSO**

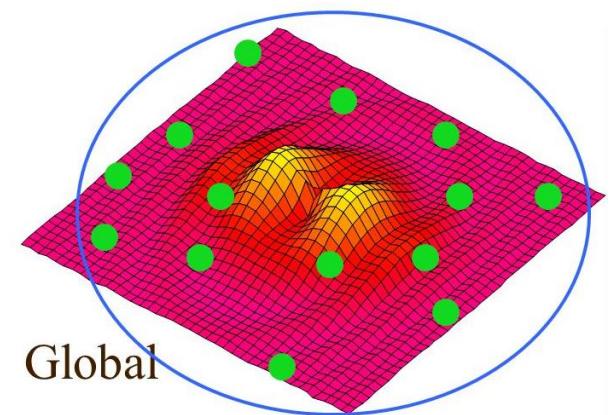
## **2. Evaluarea particulelor**

- Depinde de problema
- Functia de fitness

# Algoritm PSO

## 3. Pentru fiecare particula $x$

- Actualizarea memoriei
  - Stabilirea celei mai bune particule din swarm **gbest** / dintre particulele vecine **ibest**

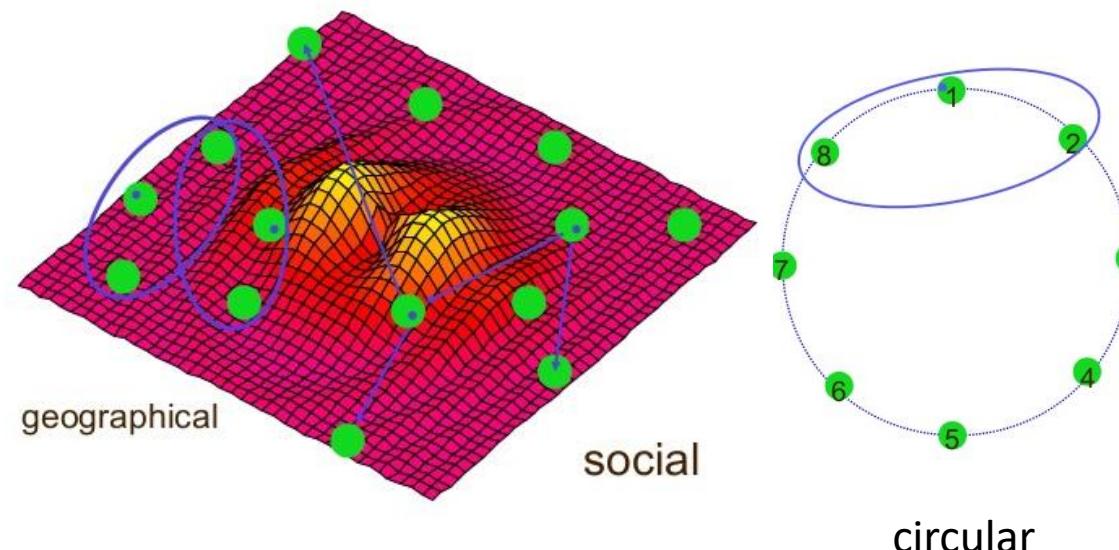


Vecinatatea unei particule:

- Globala
- Locala

Tipul vecinatatii:

- Geografica
- Sociala
- Circulara



# Algoritm PSO

## 3. Pentru fiecare particula $x$

- Actualizarea memoriei
  - Stabilirea celei mai bune particule din swarm **gbest** / dintre particulele vecine **ibest**
  - Stabilirea celei mai bune poziții (cu cel mai bun fitness) în care a ajuns până atunci – **pbest**

# Algoritm PSO

## 3. Pentru fiecare particula $x$

- Actualizarea memoriei
- Modificarea vitezei  $v$

**Eq. (a):**

$$\begin{aligned} v[] &= w * v[] + \\ &\quad c1 * \text{rand}() * (\text{pbest}[] - x[]) + \\ &\quad c2 * \text{rand}() * (\text{gbest}[] - x[]) \end{aligned}$$

- Modificarea poziției

**Eq. (b):**

$$x[] = x[] + v[]$$

$v$  – viteza particulei

$x$  – particula curenta (solutia)

$\text{rand}()$  – numar aleator intre 0 si 1

$w$  = factor de inertie

$c_1$  = factor de invatare cognitiv

$c_2$  = factor de invatare social

$w, c_1, c_2$  (ponderi pozitive)

# Pseudocod PSO

For each particle

    Initialize particle

END

Do

    For each particle

        Calculate fitness value

        If the fitness value is better than (pBest) then pBest= current value

End

gBest = the particle with the best fitness value of all the particles

For each particle

    Calculate particle velocity according equation (a)

    Update particle position according equation (b)

End

While maximum iterations or minimum error criteria is not attained

# Modificarea vitezei si pozitiei

## 3. Pentru fiecare particula $x_i = x_{i1}, \dots, x_{iD}$

- $x_{ij}$  = pozitia particulei i in dimensiunea j,  $j=1\dots D$
- $v_{ij}$  = viteza particulei i in dimensiunea j,  $j=1..D$
- Modificarea vitezei pe fiecare dimensiune j

**Eq. (a):**

$$v_{ij} = w * v_{ij} + c1 * \text{rand}() * (pbest_j - x_{ij}) + c2 * \text{rand}() * (gbest_j - x_{ij})$$

OBS: Viteza este restrictionată de o valoare maxima presetata Vmax.

- Modificarea poziției pe fiecare dimensiune j

**Eq. (b):**

$$x_{ij} = x_{ij} + v_{ij}$$

$i = 1, \dots, N$  (numarul de particule)

$j = 1, \dots, D$  (numarul de dimensiuni)

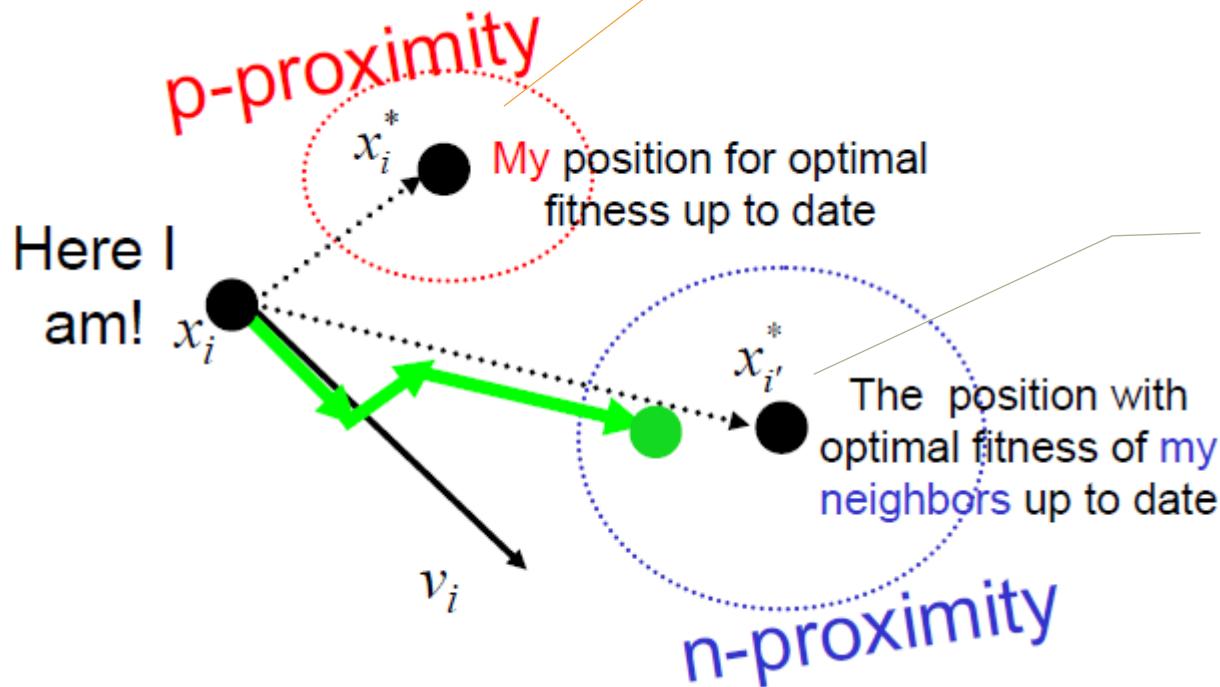
# Parametrii PSO

- **Numarul de particule (de obicei 10-50)**
- **$w : w * v_{ij}$  – termen inertial**
  - forțează particula să se deplaseze în aceeași direcție ca și până acum
  - balansează căutarea între explorare globală ( $w$  mare) și locală ( $w$  mic)
  - poate fi constantă sau descrescătoare (pe măsura „îmbătrânirii” grupului)
- **$c1: c1 * rand() * (pbest_j - x_{ij})$  – termen cognitiv**
  - forțează particula să se deplaseze spre cea mai bună poziție atinsă până atunci (tendință de conservare)
- **$c2 : c2 * rand() * (gbest_j - x_{ij})$  – termen social**
  - forțează particula să se deplaseze spre cea mai bună poziție a vecinilor; spirit de turmă, de urmăritor
- Cei doi factori  $c1$  și  $c2$  pot fi egali (de obicei  $c1=c2=2$ ) sau diferiți (de obicei  $c1 > c2$  și  $c1+c2=4$  sau  $c1+c2 < 4$ )
- Exemplu de setare:  $w=1$ ,  $c1=2$ ,  $c2=2$

# Miscarea particulelor

Componenta cognitiva

PBEST



GBEST

Componenta sociala

A. Martinoli, 2006

# Algoritmul PSO

Initializare populatie  $N$  particule ( $v, x$ )

**Repeat**

**for**  $i = 1$  to  $N$  **do**

**if**  $f(x_i) < f(x_{pbest_i})$  **then**  $x_{pbest_i} = x_i$

$x_{gbest} = \text{best}(x_{pbest})$

**for**  $j = 1$  to  $D$  **do**

        Calcul viteza  $v_{ij}$

        Calcul pozitie  $x_{ij}$

**end**

**end**

**until** STOP criterion met

# Exemplu aplicare

$$f(x) = x_1^2 + x_2^2 + x_3^2$$

- O particula este  $x = (x_1, x_2, x_3)$
- Functia de fitness este  $f(x)$
- Numarul de particule = 10
- Intervalul in care ia valori fiecare  $x$  (setat de problema): [-10,10]
- Vmax = 20
- w=1
- c1 = c2 = 2
- Conditia de oprire: numarul maxim de iteratii 2000

# PSO vs AE

- Diferente PSO fata de AE
  - Nu există un operator de recombinare directă – schimbul de informație are loc în funcție de experiența particulei și în funcție de cea a celui mai bun vecin și nu în funcție de părinții selectați pe baza fitness-ului
  - Update poziție ~ similar cu mutația
  - Nu se folosește selecția – supraviețuirea nu este legată de fitness
- Versiuni ale algoritmului de tip PSO
  - PSO binar discret
  - PSO cu mai mulți termeni de învățare socială
  - PSO cu particule eterogene
  - PSO ierarhic

# PSO discret (binar)

- Versiune a PSO pentru spațiu de căutare discret
- Poziția unei particule
  - Potențială soluție a problemei -> string binar
  - Se modifică în funcție de viteza particulei
- Viteza unei particule
  - element din spațiu continuu e.g. valoare in [0,1]
  - se modifică conform principiilor de la PSO standard
  - se interpretează ca probabilitatea de modificare a bitului corespunzător din poziția particulei

$$x_{ij} = \begin{cases} 1, & \text{daca rand} < s(v_{ij}) \\ 0, & \text{altfel} \end{cases}, \text{ unde } s(v_{ij}) = \frac{1}{1 + e^{-v_{ij}}}$$

# Remarci PSO

- Pericole
  - Particulele tind să se grupeze în același loc
  - Converg prea repede și nu reușesc să evadeze dintr-un optim local
  - **Soluția: Reinițializarea unor particule**
- Analiza algoritmilor de tip PSO
  - Indicele de dispersie
    - Măsoară gradul de împrăștiere a particulelor în jurul celei mai bune particule din grup
    - Media distanțelor absolute (pe fiecare dimensiune) între fiecare particulă și particula cea mai bună
    - Explică gradul de acoperire (întins sau restrâns) a spațiului de căutare
  - **Indicele vitezei**
    - Măsoară viteza de mișcare a grupului într-o iteratăie
    - Media vitezelor absolute
    - Explică cum (agresiv sau lent) se mișcă grupul

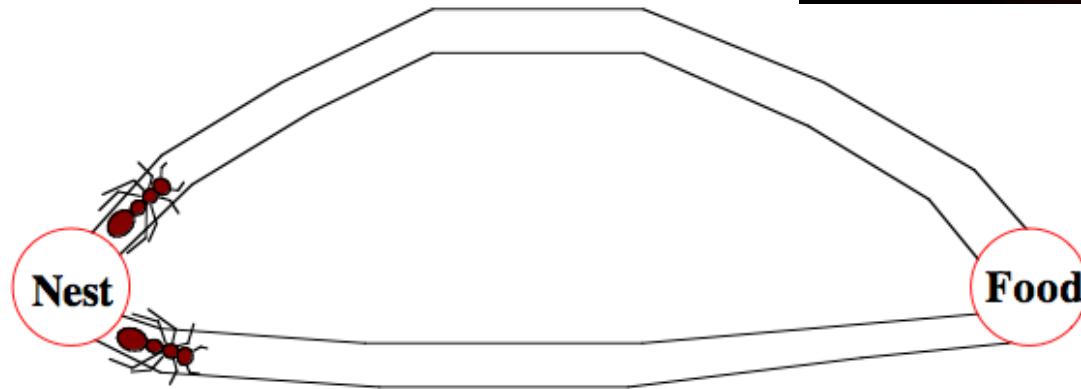
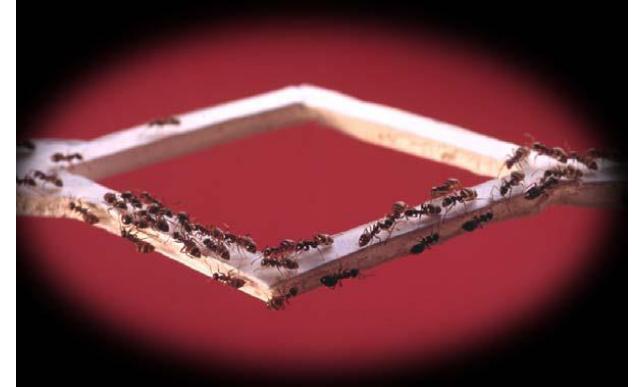
# Aplicatii PSO

- Controlul și proiectarea antenelor
- Aplicații biologice, medicale, farmaceutice
  - Analiza tremurului în boala Parkinson
  - Clasificarea cancerului
  - Predicția structurii proteinelor
- Comunicare în rețele
- Optimizare combinatorială
- Optimizări financiare
- Analiza imaginilor și analiza video
- Robotică
- Planificare
- Securitatea rețelelor, detecția intrușilor, criptografie, criptanaliză
- Procesarea semnalelor

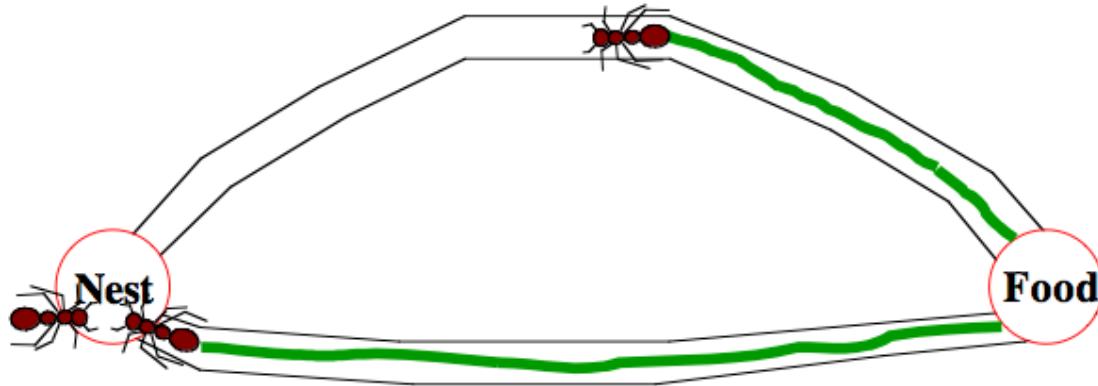
# Ant Colony Optimization (ACO)

- Marco Dorigo, 1991 (Ant System), 1995, 1999 (ACO), ...
  - Dorigo M., Maniezzo V., and Colorni A., "The Ant System: Optimization by a Colony of Cooperating Agents". *IEEE Trans. Systems, Man, and Cybernetics*, Vol. 26, pp. 29-41, 1996.
  - M. Dorigo and T. Stuetzle, "Ant Colony Optimization", MIT Press, 2004
  - M. Dorigo, M. Birattari, T. Stutzle, Ant colony optimization – Artificial Ants as a computational intelligence technique, IEEE Computational Intelligence Magazine 2006
- <http://iridia.ulb.ac.be/~mdorigo/ACO/about.html>
- Inspirata de comportamentul real al furnicilor
  - Furnicile gasesc cel mai scurt drum intre casa si sursa de hrana
  - Furnicile depun feromon pe drumul urmat – ceea ce ghideaza alte furnici in alegerea drumului (Stigmergy)
- Comportamentul real al furnicilor formalizat intr-o **metaeuristica**

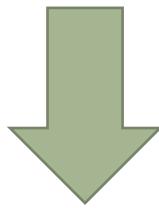
# Sursa de inspiratie...



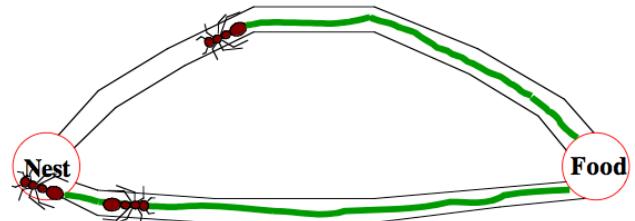
2 furnici pornesc cu probabilitate egala pe unul din  
cele 2 drumuri



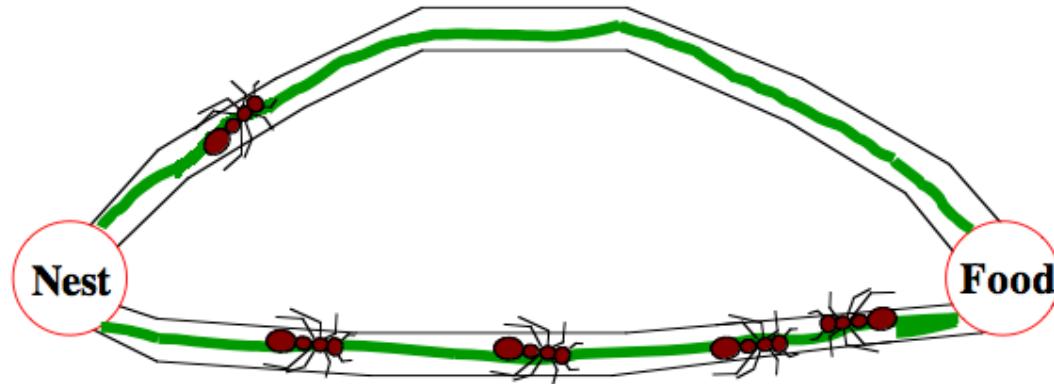
Furnica care merge pe drumul mai scurt ajunge  
mai repede inapoi la casa



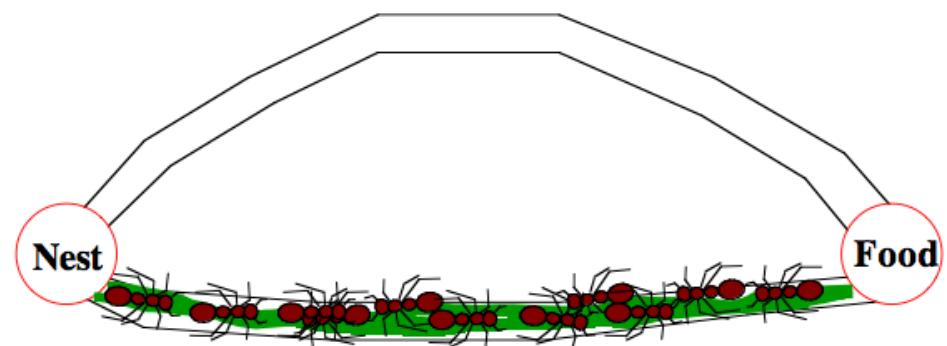
Densitate mai mare de feromon  
pe drumul mai scurt



Urmatoarea furnica alege  
drumul mai scurt

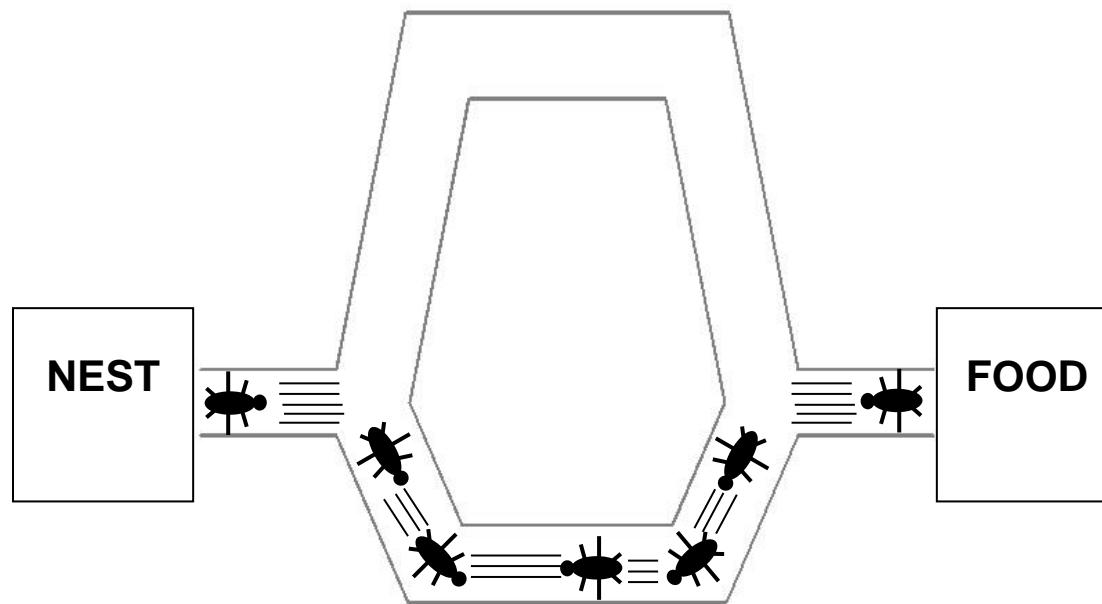


Dupa mai multe iteratii – tot mai multe furnici aleg drumul cu o cantitate mai mare de feromon - reinforcement



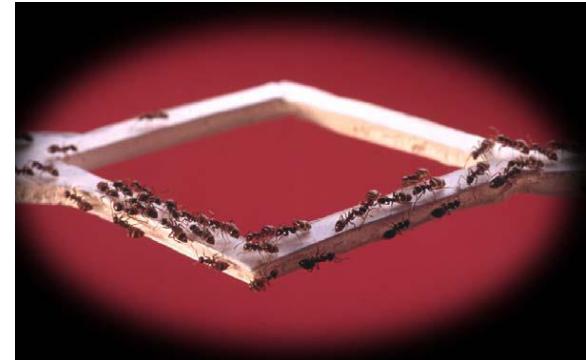
Dupa un timp – cel mai scurt drum exclusiv folosit de furnici

# Comunicare indirectă prin feromon - stigmergy

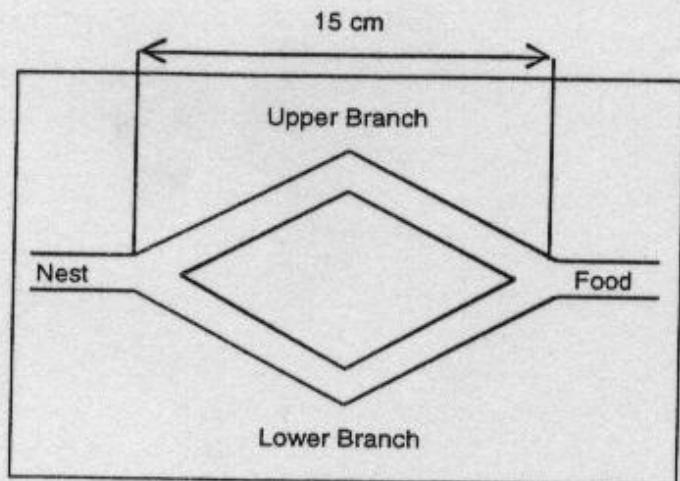


Cu cat mai multe furnici merg pe un drum cu atat acel  
drum devine mai atractiv

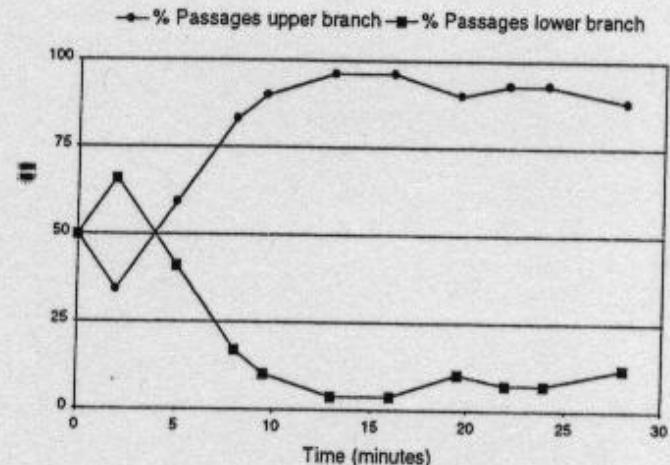
# Convergenta



Chiar si atunci cand cele 2 drumuri au lungime egala unul din ele devine preferat dupa un timp (Deneubourg et al)



(a)



(b)

Figure 1. Single bridge experiment. (a) Experimental setup. (b) Results for a typical single trial, showing the percentage of passages on each of the two branches per unit of time as a function of time. Eventually, after an initial short transitory phase, the upper branch becomes the most used. After Deneubourg et al., 1990 [25].

# ACO – aspecte teoretice

- Problema de optimizare – reprezentata ca un graf
  - Problema de optimizare trebuie transformată într-o problemă de identificare a drumului optim într-un graf orientat
- Solutie candidat  $\Leftrightarrow$  drumul construit de furnici artificiale
  - In fiecare iteratie, furnurile contruiesc solutii partiale depozitand o cantitate de feromon pe muchia traversata
- Metodă de optimizare bazată pe:
  - Colonii de furnici (în loc de cromozomi) care caută soluția optimă
  - Cooperare (în loc de competiție ca în cazul AE)
- Fiecare furnică:
  - Se mișcă și depune o cantitate de feromon pe drumul parcurs
  - Alege drumul pe care să-l urmeze în funcție de:
    - Feromonul existent pe drum
    - Informația euristică asociată aceluiași drum
  - Cooperă cu celelalte furnici prin urma de feromon de pe drum care
    - depinde de calitatea soluției și
    - se evaporă cu trecerea timpului

# ACO - algoritm

Cât timp nu s-a ajuns la nr maxim de iterații

1. Inițializare
2. Cât timp nu s-a parcurs numărul necesar de pași pentru identificarea soluției
  - *Pentru fiecare furnică din colonie*
    - Se mărește soluția parțială cu un element (furnica execută o mutare)
    - Se modifică local urma de feromon corespunzător ultimului element adăugat în soluție
3. Se modifică urma de feromon de pe drumurile parcuse de
  - Toate furnicile/cea mai bună furnică
4. Se returnează soluția găsită de cea mai bună furnică

# Ant System (AS) - Primul alg. ACO

Dorigo, Colorni, Maniezzo

- Furnicile sunt inzestrate cu o memorie in care se retin nodurile vizitate
- Solutiile sunt construite probabilistic fara a modifica feromonul in timpul constructiei unei solutii
- Furnicile depun o cantitate de feromon proportionala cu calitatea solutiei generate

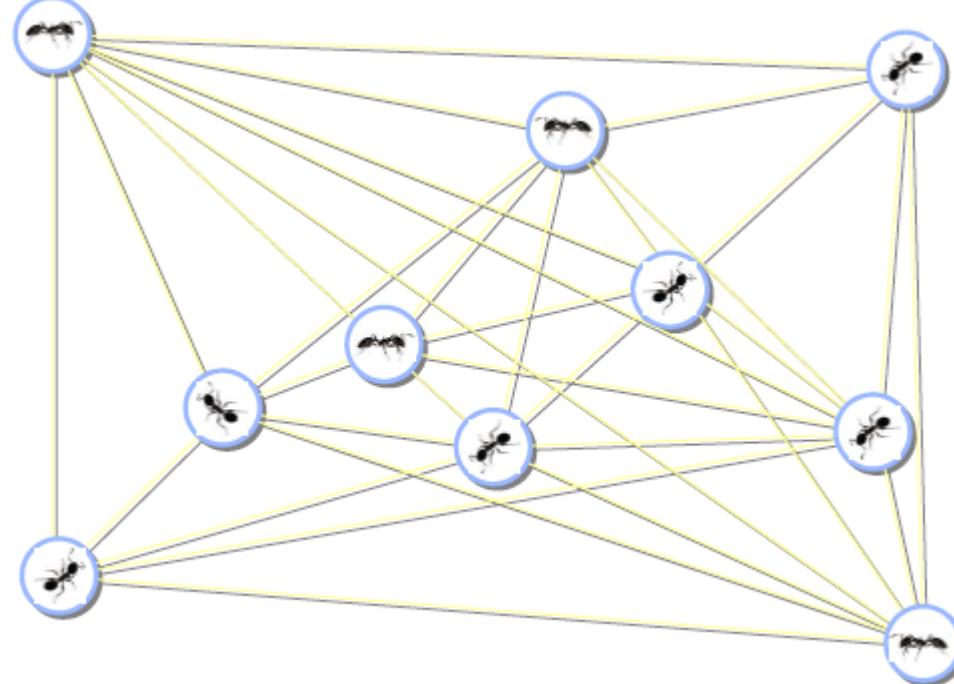
M. Dorigo, V. Maniezzo, and A. Colorni. Ant System: Optimization by a Colony of Cooperating Agents. *IEEE Transactions on Systems, Man, and Cybernetics - Part B*, 26(1):29-41, 1996.

# AS pentru TSP

$m$  – numarul total al furnicilor

La o iteratie  $t$  – un anumit numar de furnici  $b_i(t)$  in fiecare nod  $i$

$$\sum_{i=1}^n b_i(t) = m$$



Presupunere AS pentru TSP: **Graf total conectat**

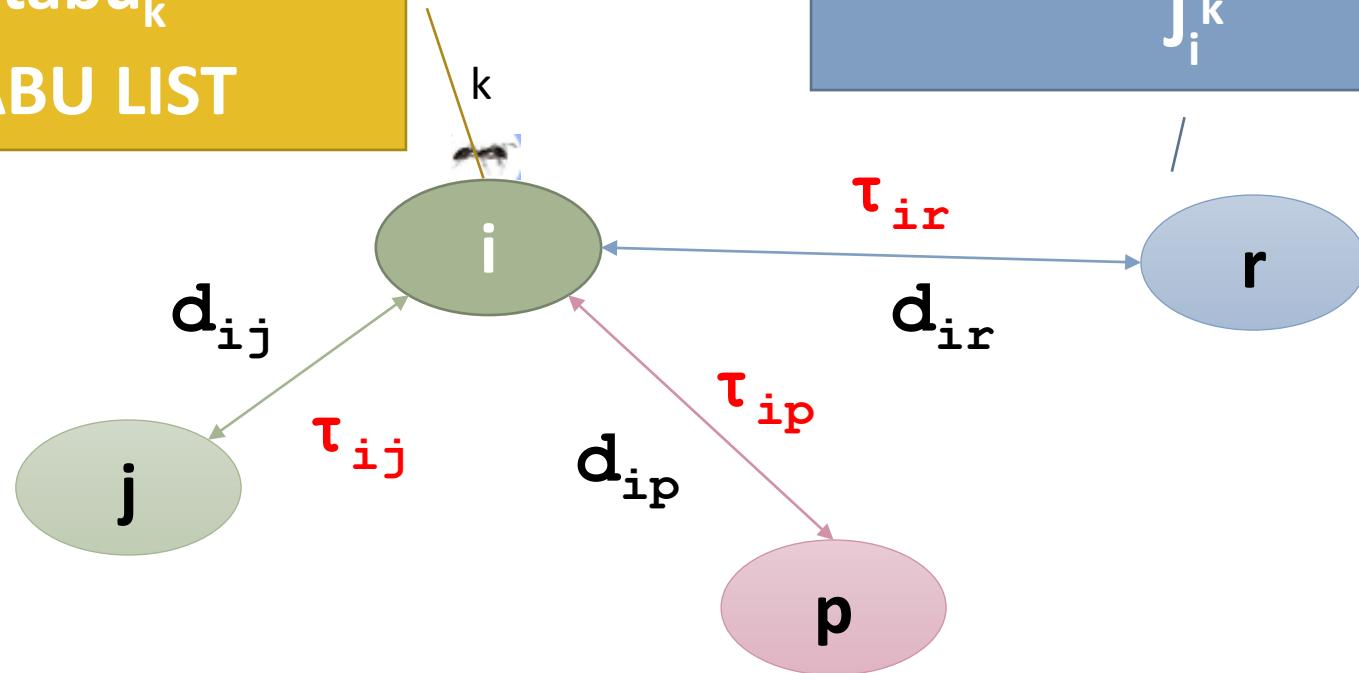
Lista nodurilor vizitate:

**tabu<sub>k</sub>**

**TABU LIST**

Lista nodurilor accesibile din i:

**J<sub>i</sub><sup>k</sup>**



$$\eta_{ij} = \frac{1}{d_{ij}} \longrightarrow$$

**Vizibilitatea** (= inversul distantei intre 2 noduri)  
 ✓ statica – informatie ce nu se schimba in timpul constructiei unei solutii

**τ<sub>ij</sub>**

Cantitatea de **feromon** virtual depusa pe arcul ce leaga nodurile i si j  
 ⇔ **Trail intensity**  
 ✓ dinamica

# Algoritmul AS pentru TSP

- **REPEAT**
  - Furnicile sunt pozitionate in noduri (n)
  - **FOR** pas = 1 to n-1 **DO**
    - **FOR** k = 1 to m **DO**
      - Furnica k alege nodul urmator aplicand regula de tranzitie (**state transition rule**)
    - **End-for**
  - **End-for**
  - Urmele de feromon sunt modificate (**pheromone trail update**)
- **UNTIL** STOP condition

# State transition rule AS

## Regula de tranzitie AS

- Regula de trecere dintr-un nod **i** într-un nod **j**
- **probabilistic**

$$p_{ij}^k(t) = \frac{[\tau_{ij}(t)]^\alpha [\eta_{ij}]^\beta}{\sum_{r \in J_i^k} [\tau_{ir}(t)]^\alpha [\eta_{ir}]^\beta}, j \notin tabu_k$$

$$p_{ij}^k(t) = 0, \text{ altfel}$$

$\alpha$  = parametru ce controleaza influenta feromonului

$\beta$  = parametru ce controleaza influenta costului (vizibilitatea)

# Pheromone trail update rule

- Regula de modificare feromon
- Aplicata dupa ce un tur complet este construit de toate furnicile

$$\tau_{ij}(t+1) = (1 - \rho)\tau_{ij}(t) + \Delta\tau_{ij}(t)$$

$$\Delta\tau_{ij}(t) = \sum_{k=1}^m \Delta\tau_{ij}^k \quad \Delta\tau_{ij}^k = \begin{cases} \frac{Q}{L_k} & , \text{daca } (i,j) \text{ apartine drumului detectat de k} \\ 0 & \text{altfel} \end{cases}$$

$\rho$  = coeficientul de evaporare feromon;  $Q$ =constanta

$L_k$  = turul detectat de o furnica  $k$

La initializare, cantitatea de feromon pe fiecare muchie =  $\tau_0$

# AS pentru TSP cu 10 orase

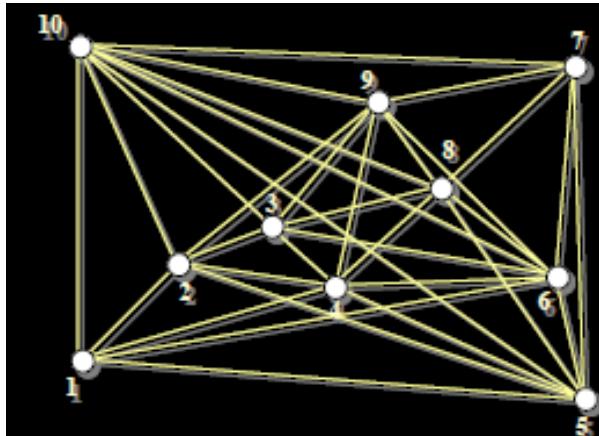
$$m = n$$

$$\alpha = 1$$

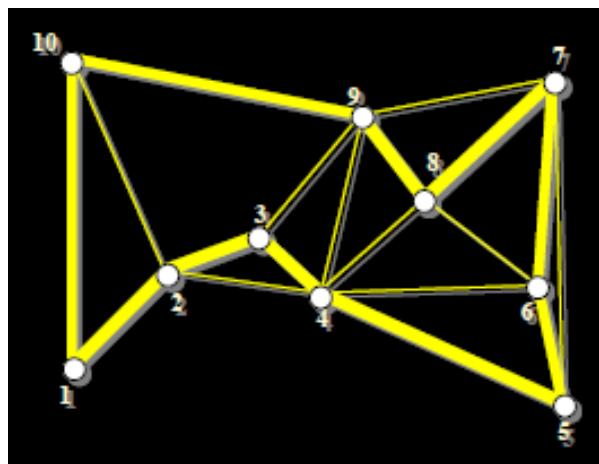
$$\beta = 1$$

$$\rho = 0.5$$

$$Q = 100$$



La initializare – cantitate de feromon egala pe toate muchiile



Dupa 100 de iteratii  
(cicluri)

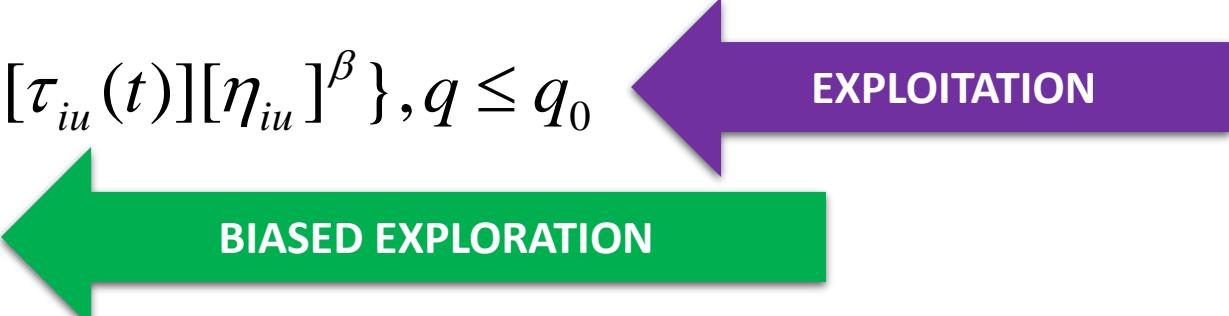
# ACS – Ant Colony System

- Im bunatatiri majore aduse algoritmului AS
- Dorigo, Gambardella, 1996, 1997
  - Dorigo and L. M. Gambardella. Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem. *IEEE Transactions on Evolutionary Computation*, 1(1):53-66, 1997
- ACS difera de AS prin 3 modificari principale
  - *Regula de tranzitie* modificata pentru un echilibru explorare/exploatare mai bun
  - Regula de modificare feromon *globala* este aplicata numai arcelor ce apartin **celui mai bun drum** detectat
  - Este introdusa si o regula de modificare feromon la nivel *local* (**local pheromone updating rule**)

# Algoritmul ACS

- **REPEAT**
  - Fiecare furnica este pozitionata intr-un nod de start
  - **REPEAT**
    - **State transition rule:** Fiecare furnica construieste incremental o solutie
    - **Local pheromone updating rule**
  - **UNTIL** Toate furnicile - un drum complet
  - **Global pheromone updating rule:** Urmele de feromon sunt modificate la nivel global
- **UNTIL** STOP condition

# Regula de tranzitie ACS

$$j = \begin{cases} \arg \max_{u \in J_i^k} \{ [\tau_{iu}(t)] [\eta_{iu}]^\beta \}, q \leq q_0 \\ J, q > q_0 \end{cases}$$


$q$  = variabila aleatoare uniform distribuita [0,1]

$q_0$  = parametru intre 0 si 1 => controleaza **exploatare vs. explorare**

$J$  – variabila aleatoare selectata conform distributiei de probabilitate (AS)

$$p_{ij}^k(t) = \frac{[\tau_{ij}(t)]^\alpha [\eta_{ij}]^\beta}{\sum_{r \in J_i^k} [\tau_{ir}(t)]^\alpha [\eta_{ir}]^\beta}, j \notin tabu_k$$

# ACS Global Updating Rule

## Regula globală de modificare feromon

- La nivel global, numai furnica ce a generat cea mai bună soluție depune feromon
  - Feromon modificat numai pe cel mai bun drum  $L^+$  generat până la iterată curentă inclusiv

$$\tau_{ij}(t+1) = (1 - \rho)\tau_{ij}(t) + \rho \Delta \tau_{ij}(t)$$

$$\Delta \tau_{ij} = \begin{cases} \frac{1}{L^+}, & (i, j) \in L^+ \\ 0 & \text{altfel} \end{cases}$$

*Diferente mici : dacă se consideră  $L^+$  ca fiind cel mai bun din iterată curentă*

# ACS Local Updating Rule

## Regula locala de modificare feromon

- In timpul construirii unei solutii – drum – furnicile depoziteaza feromon pe muchiile traversate dupa regula:

$$\tau_{ij}(t+1) = (1 - \xi)\tau_{ij}(t) + \xi\tau_0$$

- $\xi$  – parametru; ex.  $\xi = 0.1$
- $\tau_0$  – parametru; reprezinta de asemenea cantitatea initiala de feromon pe fiecare muchie



# ACS – Rezultate si Comparatii

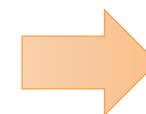
$$\alpha = 1 \quad \rho = \xi = 0.1$$

$$\beta = 2 \quad \tau_0 = (n \cdot L_{nn})^{-1}$$

$$q_0 = 0.9 \quad m = 20$$

$L_{nn}$  - nearest neighbor heuristic builds a tour by adding the closest node in terms of distance from the last node inserted in the path

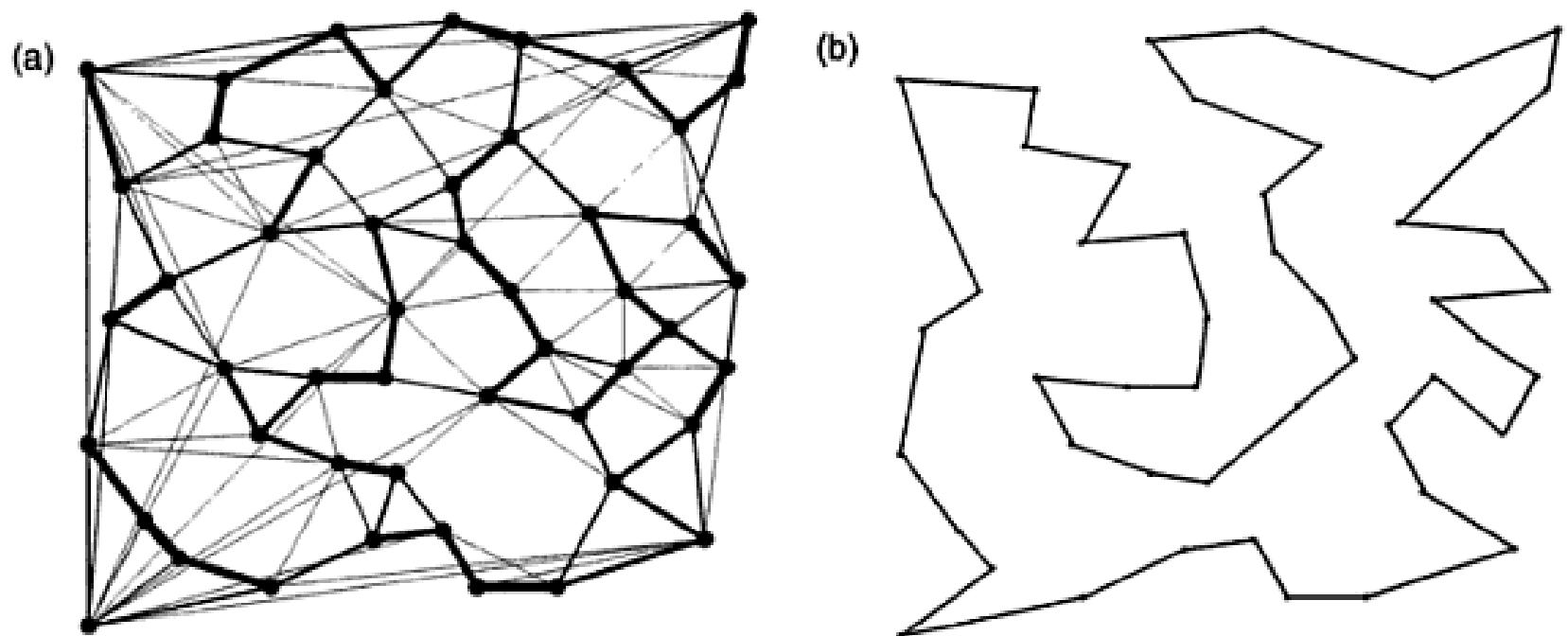
ACS – 1250  
iteratii



25000 tururi  
generate

Problema	Info	Ant Colony System (Dorigo, Gambardella, 1997)	Algoritmi Genetici (Whitley et al, 1989)
Eil50 50 orase	Best tour	<b>425</b>	<b>428</b>
	# tururi pana la best	1830	25000
Eil75 75 orase	Best tour	<b>535</b>	<b>545</b>
	# tururi pana la best	3480	80000
KroA100 100 orase	Best tour	<b>21282</b>	<b>21761</b>
	# tururi pana la best	4820	103000

# Exemplu solutie pentru Eil50



# Max-Min Ant System (MMAS)

- T. Stutzle, H. Hoos, 1997
- Im bunatatile AS
- Numai cea mai buna furnica modifica feromonul
- Cantitatea de feromon este pastrata intre anumite limite (min..max)

$$\tau_{ij} = (1 - \rho)\tau_{ij} + \Delta\tau_{ij}^{best}$$

$$\Delta\tau_{ij}^{best} = \begin{cases} \frac{1}{L_{best}} & , \text{daca } (i,j) \text{ apartine celui mai bun drum} \\ 0 & \text{altfel} \end{cases}$$

# Exemplu aplicare ACO pentru TSP

## 1. Inițializare:

- $t := 0$  (timpul)
- pentru fiecare muchie  $(i,j)$  se inițializează
- se plasează aleator  $m$  furnici în cele  $n$  noduri-oraș ( $m \leq n$ )
- fiecare furnică își modifică memoria (lista cu orașele vizitate)

$$\left\{ \begin{array}{l} \tau_{ij} = \tau_0 \\ \Delta \tau_{ij} = 0 \end{array} \right.$$

## 2. Cât timp nu s-a parcurs numărul necesar de pași pentru construcția soluției (nr de pași = $n$ )

- Pentru fiecare furnică din colonie
  - Se mărește soluția parțială cu un element (furnica execută o mutare) și fiecare furnică  $k$  (aflată în orașul  $i$ ) alege următorul oraș pe care îl vizitează (**State Transition Rule**)
  - Se modifică local urma de feromon lăsată de fiecare furnică pe ultimul element adăugat în soluție (**Local Pheromone Update Rule**)

## 3. Se modifică urma de feromon de pe: (**Global Pheromone Update Rule**)

- drumurile parcuse de toate furnicile (AS)
- cel mai bun drum (ACO)
- cel mai bun drum parcurs de cea mai bună furnică (MMAS)

# ACO - proprietăți

- Algoritm iterativ
- Algoritm care construiește progresiv soluția pe baza
  - Informațiilor euristicice
  - Urmei de feromon
- Algoritm stocastic
- Avantaje
  - Rulare neîntreruptă și adaptabilă schimbării în timp real a datelor
  - Feedback-ul pozitiv ajută la descoperirea rapidă a soluției
  - Calculul distribuit evită convergența prematură
  - Euristică greedy ajută la găsirea unei soluții acceptabile încă din primele stadii ale căutării
  - Interacțiunea colectivă a indivizilor
- Dezavantaje
  - Converge încet față de alte căutări euristicice
  - Funcționează relativ slab pentru instanțe TSP cu mai mult de 100 orașe

# Aplicatii ACO

- Problema comis-voiajorului
  - Traveling salesperson problem - TSP
- Probleme de rutare in retele (de telecomunicatii)
  - Network routing
- Problema rutarii vehiculelor
  - Vehicle Routing Problem – VRP
- Probleme de asignare
  - Quadratic Assignment Problem – QAP
- Probleme de planificare
  - Scheduling problem
  - Job-shop scheduling
  - Satellite Image Classification
  - Movie effects
- etc

# Recap

- PSO
  - Algoritm de căutare locală în fascicol
  - Potențialele soluții -> particule caracterizate prin:
    - poziție în spațiul de căutare
    - Viteză
  - Căutare cooperativă și perturbativă bazată pe
    - Poziția celei mai bune particule din grup
    - Cea mai bună poziție a particulei de până atunci (particula are memorie)
- ACO
  - Algoritm de căutare locală în fascicol
  - Potențialele soluții -> furnici caracterizate prin:
    - Memorie – rețin pașii făcuți în construirea soluției
    - Miros – iau decizii pe baza feromonului depus de celelalte furnici (comportament social, colectiv, colaborativ)
  - Căutare cooperativă și constructivă



**BABEŞ-BOLYAI UNIVERSITY**  
Faculty of Mathematics and Computer Science



# Inteligentă Artificială

*8: Modele cunoscute*

Camelia Chira

[cchira@cs.ubbcluj.ro](mailto:cchira@cs.ubbcluj.ro)

# Important: modalitatea de examinare IA

## Examen: P2 =max.550 puncte

### Evaluare:

- Prezentare = 250p (de submis pana in 9 mai, programarea prezentarilor 10 si 17 mai)
- Raport si cod sursa = 300p (de submis pana la data examenului)

Data limita alegere tema: **19 aprilie** (*se trece tema in fisierul excel din Files*)

**Seminar 5:** prezentarea preliminara (articol ales, tema, idee de modificare)

Data limita submitere prezentare: **9 mai, ora 13:00**

Programarea prezentarilor:

- **10 mai, 13:00-14:50**
- **17 mai, 13:00-14:50**



# Am vazut deja ca...

- **Algoritmi evolutivi**

- **Reprezentarea** indivizilor: Codificarea binara, Codificarea reala, Permutari
- **Functia de evaluare** sau de fitness
- Operatori de **variatie**: incruisare, mutatie
- **Selectia**
- **Initializarea**

- **Algoritmi inspirati de natura (swarm intelligence)**

- **Particle Swarm Optimization (PSO)**
- **Ant Colony Optimization (ACO)**

# Astazi...

- Selectia si managementul populatiei
- Setarea parametrilor: parameter tuning / parameter control
- Algoritmi evolutivi cunoscuti
- Versiuni consacrate istoric
  - Algoritmi genetici
  - Strategii evolutive
  - Programare evolutiva
  - Programare genetica
- Modele
  - Optimizare numerica
  - Optimizare cu restrictii
  - Optimizare multimodala
  - Optimizare multicriteriala

# Selectia si managementul populatiei

- Modele de management a populatiei
- Operatori de selectie
- Pastrarea diversitatii (Diversitate = numarul de fitnesses /phenotypes /genotypes diferite)

Marim diversitatea  
populatiei prin *operatori  
de variatie*

- Incrusicare
- Mutatie

=> **novelty**

Micsoram diversitatea  
populatiei prin *selectie*

- Parinti
- Generatia urmatoare

=> **quality**

# Managementul populatiei: modele

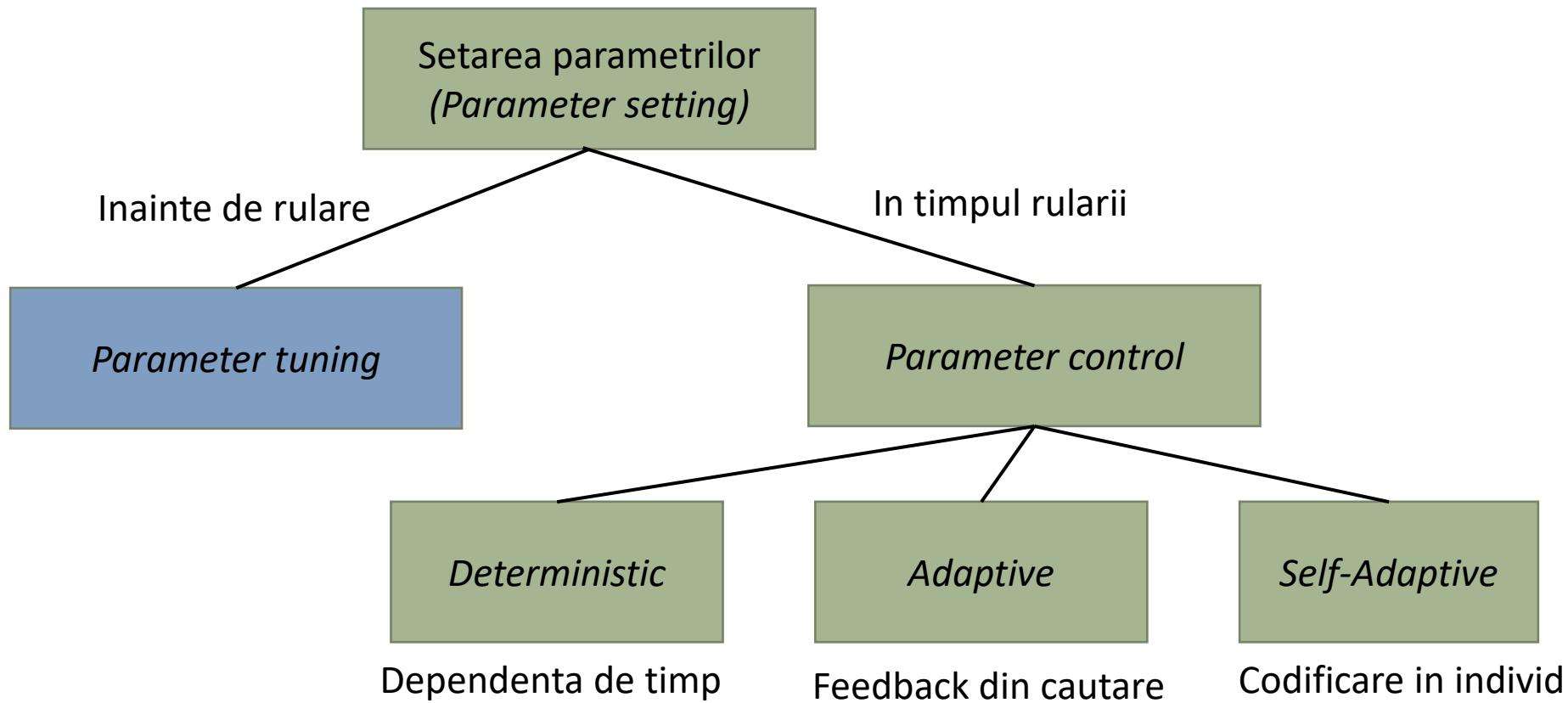
- **Model generational** (generational model)
  - Fiecare individ supravietuieste exact o generatie
  - Toti parintii sunt inlocuiti de descendenti
- **Model steady-state** (steady-state model)
  - Un offspring este creat intr-o generatie
  - Un individ din populatie este inlocuit
- Generation Gap
  - Proportia din populatie inlocuita
  - 1.0 pentru model generational
  - $1/\text{pop\_size}$  pentru model steady-state

# Managementul populatiei

- Competitie pe baza fitnessului
  - Selectia parintilor: selectia individelor din generatia curenta pentru variatie
    - Fitness-proportionate selection
    - Rank-based selection
    - Tournament selection
    - Uniform selection
  - Selectia supravietuitorilor: selectie din parinti si offspring pentru generatia urmatoare
    - Age-based selection
    - Fitness-based replacement
  - Selectia opereaza asupra unui individ intreg (nu depinde de reprezentare)

# Setarea parametrilor AE

- Gasirea celor mai bune valori pentru toti parametrii unui AE este o problema complexa



# Parameter tuning (*inainte de rulare*)

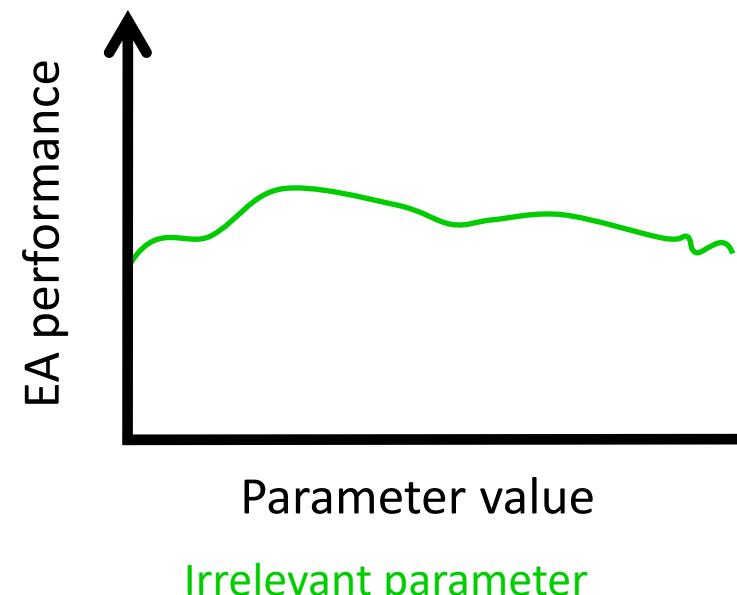
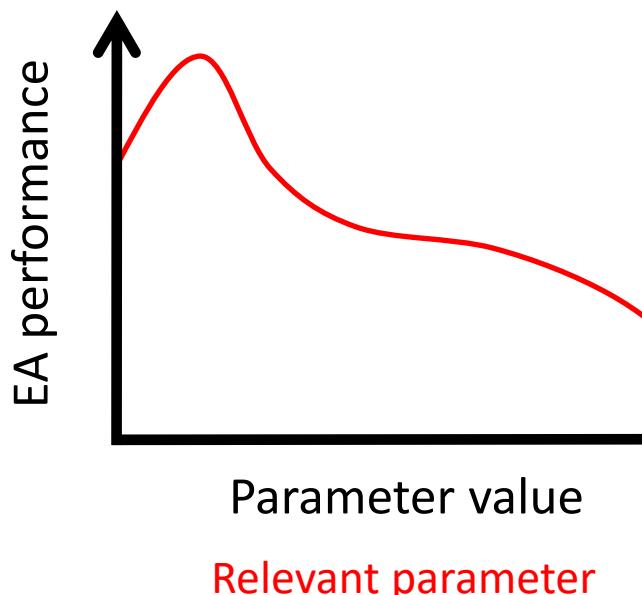
- Testam si comparam diferite valori inainte de rularile “reale”
- Probleme
  - Greseli in setari pot insemana erori mari si performanta suboptimala
  - Consuma timp
  - Parametrii nu sunt independenti unul de altul (ci interactioneaza); o cautare exhaustiva pentru toate valorile posibile si combinatiile lor nu este posibila
  - Valori bune se pot dovedi slabe in timpul rularii
- Avantaje
  - Mai usor, nevoie imediata rezolvata
  - Poate si strategiile de control au parametrii
  - Cunostinte despre spatiul de cautare pot ajuta
  - Exista indicatii ca o setare corecta si buna a parametrilor merge mai bine decat controlul parametrilor

# Testarea unor valori de parametri

- Rulam AE cu parametrii alesi pentru problema data
- Notam performanta AE in acea rulare:
  - **Calitatea solutiei** – cel mai bun fitness la terminare
  - **Viteza** – timpul necesar sa ajungem la o solutie de anumita calitate
- AEs sunt algoritmi stocastici => sunt necesare repetari pentru o evaluare relevanta
  - **Performanta medie**: a fitnessului (*Mean Best Fitness -MBF*), vitezei, AES – *Average number of Evaluations to Solution*
  - **Rata de succes (Success Rate)**: cat la suta din rulari se termina cu succes
  - **Robustete**: variatia performantei medii in probleme diferite
- **Cea mai mare problema: cate repetitii ale testului?**

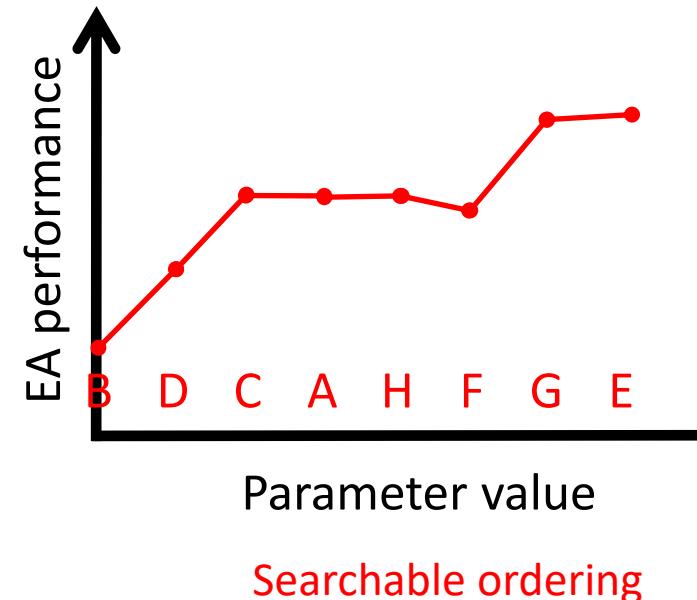
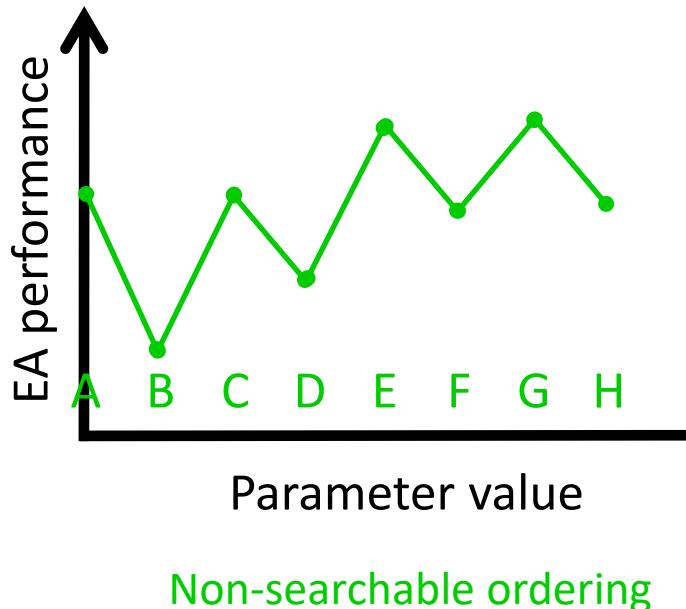
# Testarea parametrilor numerici

- Marimea populatiei, probabilitatea de incrusicare/mutatie, marimea turnirului (pentru tournament selection), ...
- Domeniu este submultime R sau N (finit sau infinit)



# Testarea parametrilor simbolici

- Operatorul de incruisare, strategie elitista, metoda de selectie,...
- Domeniu finit:{incruisare cu 1 punct de taietura, uniforma}, {Y, N}
- Non-searchable, must be sampled



# Parameter control

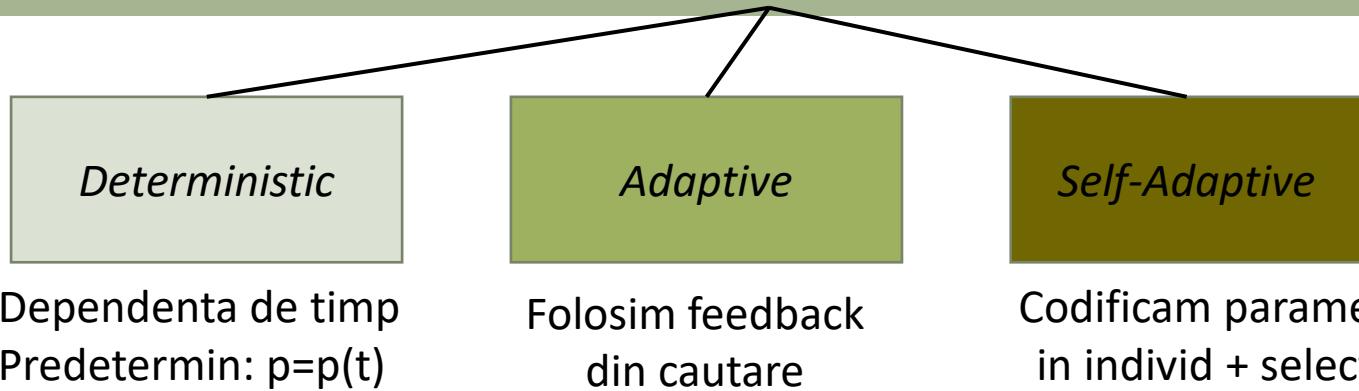
- Setarea valorilor on-line, in timpul rularii
- Motivatie: AE are multi parametri strategici
  - Operator de mutatie si rata de mutatie
  - Operator de incruisare si rata de incruisare
  - Mecanismul de selectie si presiunea de selectie (e.g. tournament size)
  - Marimea populatiei
- Cum sa gasim valori bune pentru parametrii AE?
- Parametrii AE sunt constanti desi AE este un process dinamic si adaptiv => valorile optime ale parametrilor pot varia in timpul rularii
  - Cum sa variem aceste valori?

# Parameter control

*Metodele care schimba valoarea parametrilor in timpul rularii vizeaza:*

- **Ce** se schimba?
  1. Reprezentare
  2. Functia de evaluare
  3. Operatorii de variație
  4. Selectia
  5. Populatia
- **Cum** se schimba?
  - **Determinist**: Regula modifica un parametru determinist fara feedback din cautare; dependenta de timp (regula este aplicata la un numar de generatii)
  - **Adaptiv**: Exista feedback din cautare si este folosit pentru a determina directia / amplitudinea schimbării parametrilor
  - **Auto-adaptiv**: *Evolution of evolution* (parametrii adaptati sunt codificati in structura unui individ si sunt supusi incrusiarii si mutatiei)

# Parameter control



## Probleme

- Gasirea functiei optime  $p$  este dificila, gasirea  $p(t)$  este si mai dificila
- Mecanism de feedback din cautare definit de utilizator, cum putem optimiza?
- Selectia naturala poate alege parametrii strategici?

# Parameter control: Exemplu

- Variem pasul mutatiei

## Problema:

- $\min f(x_1, \dots, x_n)$
- $L_i \leq x_i \leq U_i, i = 1 \dots n$  (limite)
- $g_i(x) \leq 0, i = 1 \dots q$  (restrictii de inegalitate)
- $h_i(x) = 0, i = q + 1 \dots m$  (restrictii de egalitate)

## Algoritm:

- AE cu reprezentare reala  $(x_1, \dots, x_n)$
- Incrucisare aritmetica medie
- Mutatie gaussiana:  
 $x'_i = x_i + N(0, \sigma)$ ,  
deviatia standard  $\sigma$  se numeste pasul mutatiei

- AE cu reprezentare reala ( $x_1, \dots, x_n$ )
- Mutatie gaussiana:  $x'_i = x_i + N(0, \sigma)$ ,

Inlocuim constanta  $\sigma$  cu o functie  $\sigma(t)$ :

$$\sigma(t) = 1 - 0.9 \times \frac{t}{T},$$

$0 \leq t \leq T$  este numarul generatiei curente

- schimbari in  $\sigma$  sunt independente de cautare
- $\sigma$  poate fi complet predeterminat
- un anumit  $\sigma$  actioneaza asupra intregii populatii

- schimbari in  $\sigma$  se bazeaza pe feedback din cautare
- $\sigma$  nu poate fi predeterminat
- un anumit  $\sigma$  actioneaza asupra intregii populatii

Inlocuim constanta  $\sigma$  cu o functie  $\sigma(t)$  care se schimba o data la  $n$  pasi folosind regula de succes 1/5:

$$\sigma(t) = \begin{cases} \frac{\sigma(t-n)}{c}, & \text{daca } p_s > 0.2 \\ \sigma(t-n) \cdot c, & \text{daca } p_s < 0.2 \\ \sigma(t-n), & \text{altfel} \end{cases}$$

- schimbari in  $\sigma$  sunt rezultatul selectiei naturale
- $\sigma$  poate fi predeterminat
- un anumit  $\sigma$  actioneaza asupra unui singur individ

Fiecare individ are un  $\sigma$  care este codificat in individ  
Reprezentare:  $(x_1, \dots, x_n, \sigma)$

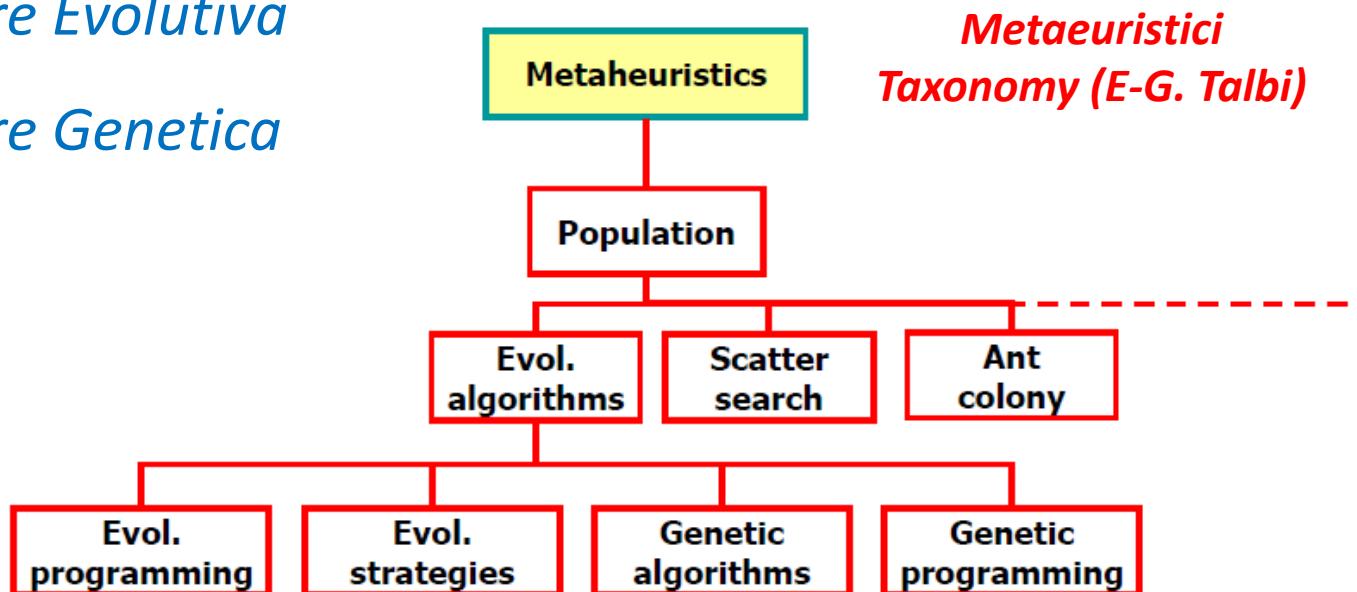
Aplicam operatorii de variatie la nivel x si  $\sigma$ :

$$\sigma' = \sigma \times e^{N(0, \sigma)}$$

$$x'_i = x_i + N(0, \sigma')$$

# Algoritmi Evolutivi

- *Algoritmi Genetici*
- *Strategii Evolutive*
- *Programare Evolutiva*
- *Programare Genetica*



# Algoritmi Genetici

- J. Holland, K. DeJong, D. Goldberg
- Holland's orginal GA – known as the simple genetic algorithm (SGA)
- *Reprezentare*: binara
- *Recombinare*: 1-point, uniforma
- *Mutatie*: bit flip cu probabilitate fixa
- *Selectia parintilor*: propotionala cu fitness (Roulette Wheel impl.)
- *Selectia supravietuitorilor*: toti urmasii inlocuiesc parintii
- **Folosit in comparatii**
- **Are multe defecte e.g. reprezentare restrictiva, operatori de variatie aplicabili numai pentru codificare binara si intreaga, selectia nepotrivita pentru populatii cu valori apropiate de fitness, model generational in loc de selectia explicita a supravietuitorilor**

# Strategii Evolutive (Evolution Strategies - ES)

I. Rechenberg, H.P. Schwefel

Reprezentare	Vectori cu valori reale
Incrucisare	Discreta sau convexa
Mutatia	Perturbatii gaussiene
Selectia parintilor	Uniform aleatoare
Selectia supravietuitorilor	$(\mu, \lambda)$ sau $(\mu + \lambda)$

- ✓ Tipic folosita in optimizarea numerica
- ✓ Self-adaptation of mutation parameters

# ES: Pseudocod (exemplu)

Minimizeaza  $f: \mathcal{R}^n \rightarrow \mathcal{R}$

$$x = \{x_1, x_2, \dots, x_n\}$$

- $t = 0$
- Crearea unui punct initial  $x^t = \{x_1^t, \dots, x_n^t\}$
- REPETA PANA CAND (*TERMIN.COND* =true)
  - Alege  $z_i$  dintr-o distributie normala pentru toti  $i = 1, \dots, n$ 
    - $y_i^t = x_i^t + z_i$
    - DACA  $f(x^t) < f(y^t)$  ATUNCI  $x^{t+1} = x^t$   
ALTFEL  $x^{t+1} = y^t$
  - $t = t+1$

# Programare Evolutiva (Evolutionary Programming - EP)

D. Fogel

Reprezentare	Vectori cu valori reale
Incrucisare	Nu se foloseste
Mutatia	Perturbatii gaussiene
Selectia parintilor	Determinista
Selectia supravietuitorilor	Probabilistic ( $\mu+\mu$ )

# EP pentru optimizarea functiilor

- Initial: machine learning tasks (masini instruibile) folosind automate cu stari finite ( $\text{inteligenta} \Leftrightarrow \text{comportament adaptiv}$ )
- EP – folosit si in probleme de cautare/optimizare
- Puncte comune cu strategiile evolutive

- Reprezentare: un cromozom are doua parti
  - Variabile:  $x_1, \dots, x_n$
  - Pasi de mutatie:  $\sigma_1, \dots, \sigma_n$
- Cromozom:  $\langle x_1, \dots, x_n, \sigma_1, \dots, \sigma_n \rangle$
- Selectia parintilor: fiecare individ creaza un descendant prin mutatie
- Recombinare: Nu se aplica

# Algoritmul EP

P1.  $t=0$ ; Initializare  $P(0)$

P2. Se evaluateaza populatia  $P(t)$

P3. Cat timp (TERM. = fals) executa

$P'(t)$ =mutatie asupra lui  $P(t)$

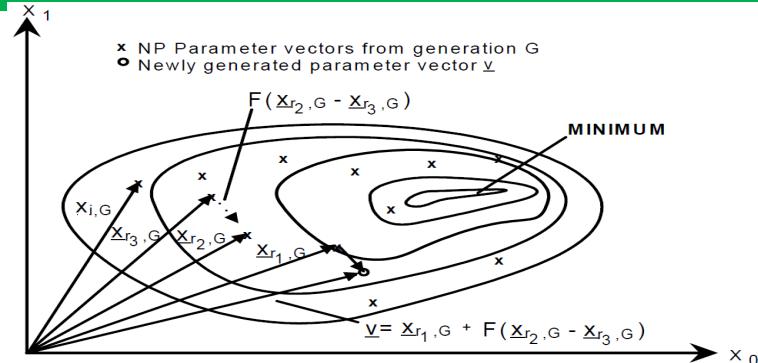
Evalueaza  $P'(t)$

$P(t+1)$ =selectie asupra  $P(t) \cup P'(t)$

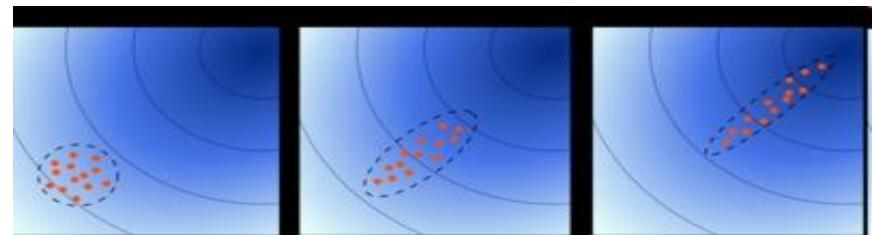
$t=t+1$

# Modele Evolutive

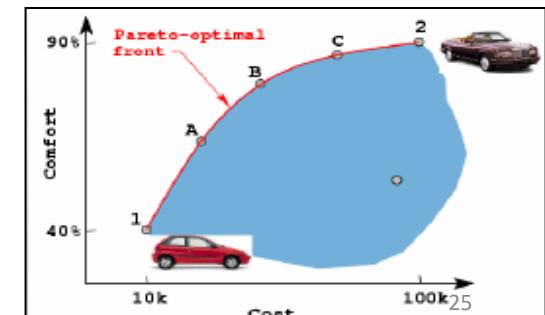
- Genetic Algorithms
- Differential Evolution (DE)
  - Numerical optimization



- CMA-ES
  - Increase the probability of successful mutation step

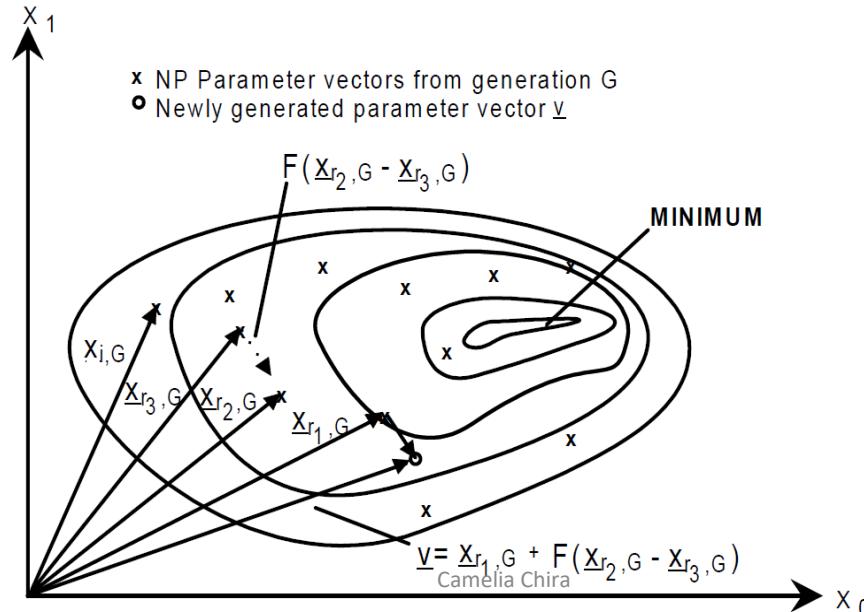


- NSGA (Non-dominated Sorting GA)
  - Multi-objective optimization



# Differential Evolution (DE)

- Storn & Price (1995)
- Optimizare numérica
- Specific
  - Mutatia - perturbatie calculata ca diferența a două elemente aleatoare ale populației
  - Patru parinti sunt necesari pentru a crea un nou individ



# Algoritmul DE

- Populatie cu **m** elemente,  $(x_1, \dots, x_m)$
- In fiecare generatie, pentru fiecare element al populatiei  $x_i$ 
  - Se selecteaza aleator trei elemente distincte ale populatiei:  $x_{r1}, x_{r2}$  si  $x_{r3}$  (parinti)
  - Se construieste un vector mutant:  
$$x'_i = x_{r1} + F(x_{r2} - x_{r3}),$$
   
     $0 < F < 2$  un factor de scala (influenteaza puterea de explorare a algoritmului) – **mutatie**
  - Se construieste un vector candidat  $y_i$  combinand componente ale lui  $x_i$  si ale lui  $x'_i$  (aceasta este etapa de **incrucisare**)
  - Se compara  $x_i$  si  $y_i$  in raport cu functia obiectiv si se selecteaza cel mai bun pentru supravietuire in generatia viitoare

**DE CANONIC - proces evolutiv de tip generational:**

*la fiecare iteratie se construieste o noua populatie aplicand prelucrarile de mai sus tuturor elementelor din populatia curenta*

# Variante DE

- DE/a/b/c
  - **a** specifica modul de alegere al primului termen din expresia mutantului (asa numitul element donor)  $x'_i = x_{r1} + F(x_{r2} - x_{r3})$
  - **b** specifica numarul de termeni de tip differenta
  - **c** specifica tipul de incruisare utilizat

## DE-a

- *rand* (DE/rand/b/c)
  - $x_{r1}$  se alege aleator din populatie
- *best* (DE/best/b/c)
  - $x_{r1}$  se alege ca fiind cel mai bun element al populatiei (notat cu  $x^*$ )
- *current-to-best* (DE/current-to-best/b/c)
  - donorul este o combinatie liniara dintre elementul curent si cel mai bun element al populatiei
  - mutantul este astfel  $x'_i = x_i + F_1(x^* - x_i) + F_2(x_{r1} - x_{r2})$

# Variante DE

- DE/a/b/c
  - **a** specifica modul de alegere al primului termen din expresia mutantului (asa numitul element donor)  $x'_i = x_{r1} + F(x_{r2} - x_{r3})$
  - **b** specifica numarul de termeni de tip differenta
  - **c** specifica tipul de incruisare utilizat

DE-b

- Numarul de termeni differenta
- Cel mai frecvent se utilizeaza o singura differenta
  - $x'_i = x_{r1} + F(x_{r2} - x_{r3})$
- Doi termeni
  - $x'_i = x_{r1} + F_1(x_{r2} - x_{r3}) + F_2(x_{r4} - x_{r5})$ .

# Variante DE

- DE/a/b/c
  - **a** specifica modul de alegere al primului termen din expresia mutantului (asa numitul element donor)  $x'_i = x_{r1} + F(x_{r2} - x_{r3})$
  - **b** specifica numarul de termeni de tip differenta
  - **c** specifica tipul de incruisare utilizat

## DE-c

- Specifica operatorul de incruisare
- Initial: **incruisarea exponentiala (exp)**
  - In noul candidat se transfera o succesiune de q componente circular consecutive din mutant, incepand de la o pozitie aleasa aleator
  - Probabilitatea de selectie a valorii q = CR<sup>q</sup> (motivul pentru care acest tip de incruisare este denumit incruisare exponentiala)
  - Similara cu incruisarea cu puncte de taietura
- **Incruisarea binomiala (bin)**
  - Similara cu incruisarea uniforma

# Variante DE

- Algoritmul DE canonic este considerat a fi **DE/rand/1/bin**
- Nu există o varianta care să fie net superioară celorlalte
- Dintre metodele care se comportă bine în raport cu multe probleme test sunt **DE/rand/1/bin** și **DE/best/1/bin**

**DE: evolving picture**  
**Darth Vader**

- Individual represented as 18000 values in range [0,1]
- Pop size = 400, F = 0.1, Crossover rate = 0.1
- Fitness = squared difference value individual - target value

**Evolving:**  
**Darth Vader**

# Inver-over Algorithm

- AE pentru abordarea TSP (Michalewicz, 1998)
- Fiecare individ intra in competitie numai cu descendantul sau
- Exista un singur operator de variatie numit **inver-over** care este adaptiv
- Acest operator este aplicat de un numar variabil de ori unui individ in timpul unei generatii
- Putem privi acest AE ca si un set de proceduri HC paralele (in spiritul algoritmului Lin-Kernighan)
- Operatorul inver-over are urmatoarele componente adaptive:
  - Numarul de inversiuni aplicate unui singur individ
  - Segmentul ce va fi inversat este determinat de un alt individ generat aleator

# Inver-over Algorithm

```
random initialization of the population  $P$ 
while (not satisfied termination-condition) do
begin
    for each individual  $S_i \in P$  do
        begin
             $S' \leftarrow S_i$ 
            select (randomly) a city  $c$  from  $S'$ 
            repeat
                if ( $\text{rand}() \leq p$ )
                    select the city  $c'$  from the remaining cities in  $S'$ 
                else
                    select (randomly) an individual from  $P$ 
                    assign to  $c'$  the city ‘next’ to the city  $c$  in the selected individual
                if (the next city or the previous city of city  $c$  in  $S'$  is  $c'$ )
                    exit from repeat loop
                invert the section from the next city of city  $c$  to the city  $c'$  in  $S'$ 
                 $c \leftarrow c'$ 
            if ( $\text{eval}(S') \leq \text{eval}(S_i)$ )
                 $S_i \leftarrow S'$ 
        end
```

# Inver-over Algorithm: Ilustrarea unei iteratii

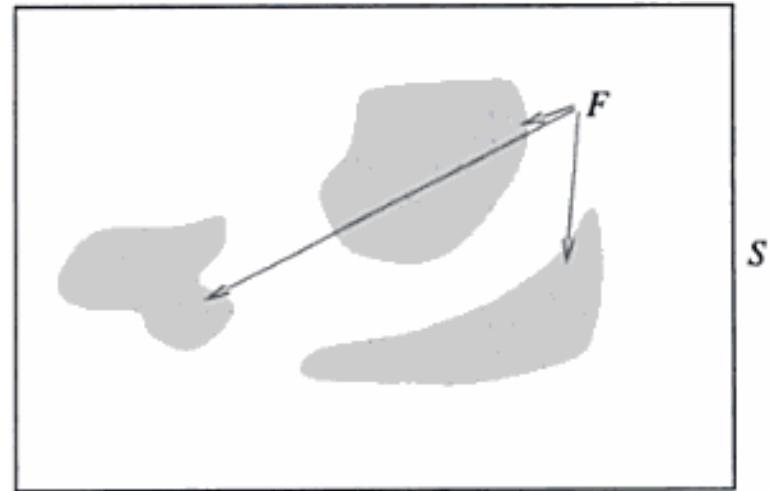
- $S' = (2 \ 3 \ 9 \ 4 \ 1 \ 5 \ 8 \ 6 \ 7)$
- Current city  $c = 3$
- Daca  $\text{rand}() \leq p$ , un alt oras  $c'$  este ales din acelasi individ  $S' \rightarrow c' = 8$ 
  - Segmentul intre  $c$  si  $c'$  este inversat  $\Rightarrow S' = (2 \ 3 \ 8 \ 5 \ 1 \ 4 \ 9 \ 6 \ 7)$
- Daca  $\text{rand}() > p$ , un alt individ este selectat aleator  $\rightarrow (1 \ 6 \ 4 \ 3 \ 5 \ 7 \ 9 \ 2 \ 8) \Rightarrow c' = 5$  (orasul din individul selectat care este dupa  $c$ )
  - Segmentul intre  $c$  si  $c'$  este inversat  $\Rightarrow S' = (2 \ 3 \ 5 \ 1 \ 4 \ 9 \ 8 \ 6 \ 7)$
  - In acest caz (3 5) este din al doilea parinte!
- **Operatorul este aplicat de mai multe ori inainte de orice evaluare!**
- Procesul se termina cand  $c'$  (dintr-un individ selectat aleator sau din individual curent) este acelasi cu urmatorul oras dupa  $c$  in individul curent
  - $S' = (9 \ 3 \ 6 \ 8 \ 5 \ 1 \ 4 \ 2 \ 7)$  si  $c = 6 \Rightarrow$  cand  $c' = 8$  procesul se termina

# Inver-over Algorithm: Observatii

- Este un algoritm rapid cu rezultate bune (obtinute in timp mai scurt decat AE bazati pe incrusisare)
- Algoritmul are putini parametri: marimea populatiei, probabilitatea  $p$  de a genera inversie aleatoare si numarul de generatii
- Operatorul inver-over combina caracteristici ale mutatiei (inversiunea) si incrusisarii
- Probabilitatea  $p$  (setata in experimente la 0.02) determina proportia de inversiuni oarbe si inversiuni ghidate (adaptive)

# Optimizare cu restrictii

- Tratarea restrictiilor incadrata in schema generala a AE



- ❖ Cum ar putea fi comparate 2 solutii ne-valide (nefezabile)?
- ❖ Care este legatura dintre functiile de evaluare a solutiilor valide vs nevalide?
- ❖ Solutiile ne-valide ar trebui considerate daunatoare si eliminate din populatie?  
Sau ar trebui ‘reparate’? Penalizate?

# Abordari optimizare numérica cu restrictii

- Metode bazate pe pastrarea fezabilitatii solutiilor
  - Folosirea unor operatori speciali
  - Cautarea in limitele regiunilor fezabile
- Metode bazate pe functii de penalizare
  - Penalizari statice/dinamice
  - Metode annealing, cu adaptare, etc
- Metode care fac o distinctie clara intre solutii valide si solutii ne-valide
  - Repararea indivizilor ne-valizi
- Metode bazate pe decodori
- Alte metode hibride

# Metoda functiilor de penalizare

- Modificarea functiei criteriu prin adaugarea unor termeni suplimentari care modeleaza restrictiile
- Exemplu

$$\left\{ \begin{array}{l} f : R^n \rightarrow R \\ f(x) \rightarrow \min \\ s.t. \\ g_i(x) = 0; i = 1, 2, \dots, m \\ h_j(x) \geq 0; j = 1, 2, \dots, k \end{array} \right.$$

$$t(a) = \begin{cases} a, & a < 0 \\ 0, & a \geq 0 \end{cases}$$

$$P : R^n \rightarrow R$$

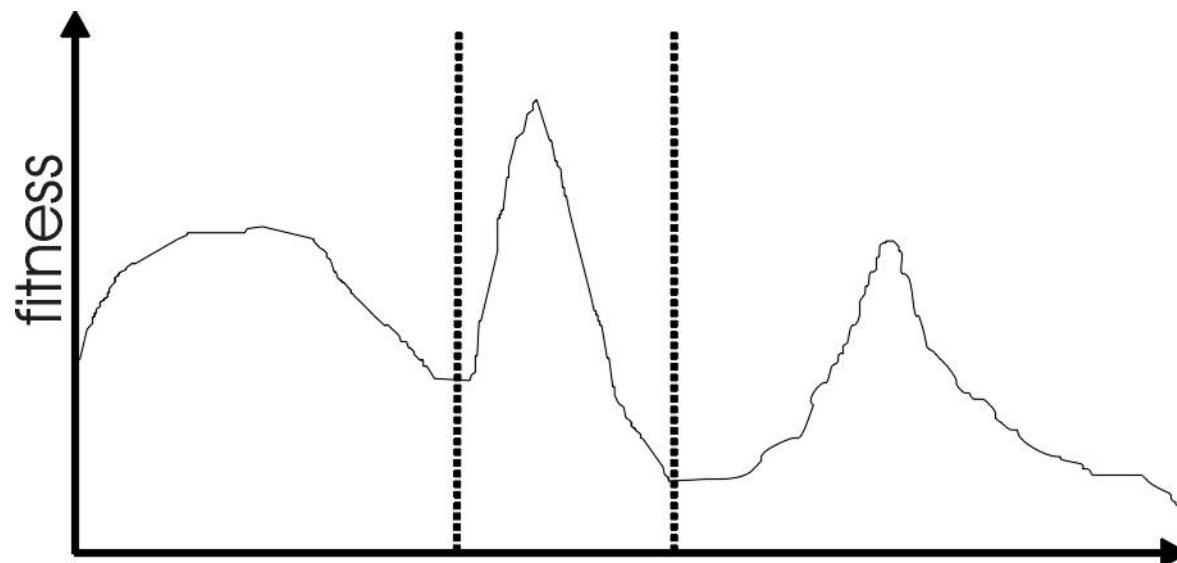
$$P(x) = f(x) + \sum_{i=1}^m (g_i(x))^2 + \sum_{j=1}^k [t(h_j(x))]^2$$

$$P(x) \rightarrow \min$$

$$\frac{1}{1+P(x)} \rightarrow \max$$

# Optimizare multimodala

- Mai multe optime locale



# Abordari explicite

- Indivizi similari intra in competitie directa prin fitness
- Indivizi similari concureaza pentru supravietuire

## Fitness Sharing

- Numarul de indivizi alocati unei nise este proportional cu valoarea fitness
- Marimea unei nise  $\sigma$
- Algoritmul evolutiv trebuie sa includa dupa fiecare generatie

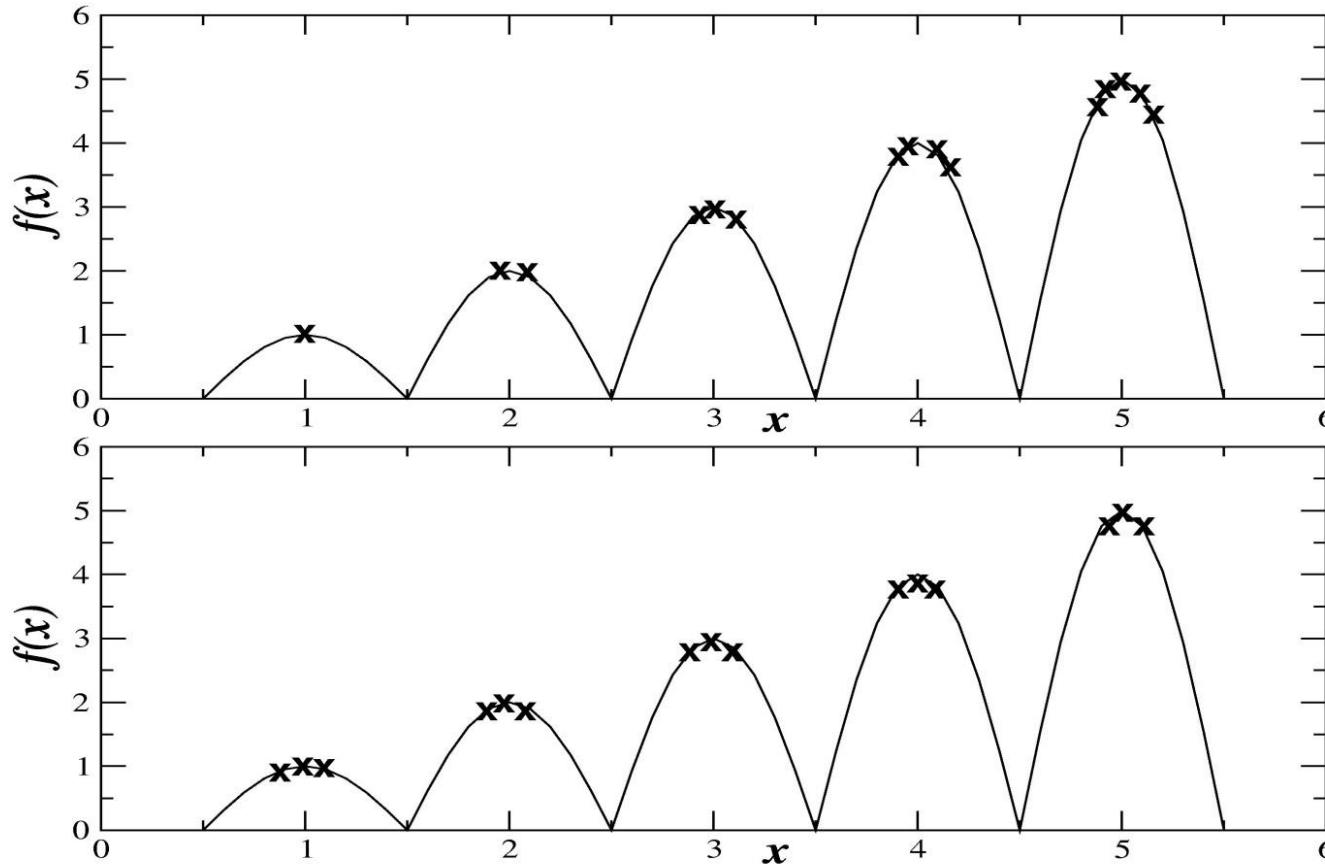
$$f'(i) = \frac{f(i)}{\sum_{j=1}^{\mu} sh(d(i, j))}$$

$$sh(d) = \begin{cases} 1 - d / \sigma & d < \sigma \\ 0 & otherwise \end{cases}$$

## Crowding

- Indivizii sunt distribuiti egal intre nise
- Foloseste distanta intre 2 indivizi ca metrica in spatiul de cautare
- Parinti alesi aleator => 2 descendenti
- Turnir parinti vs. descendenti astfel incat distantele inter-turnir sunt minime:
  - Parinti: p1, p2
  - Descendenti (offspring): o1, o2
  - $d(p1, o1) + d(p2, o2) < d(p1, o2) + d(p2, o1)$
  - o1 intra in competitie cu p1
  - o2 intra in competitie cu p2

# Fitness sharing vs Crowding



# Abordari implicate

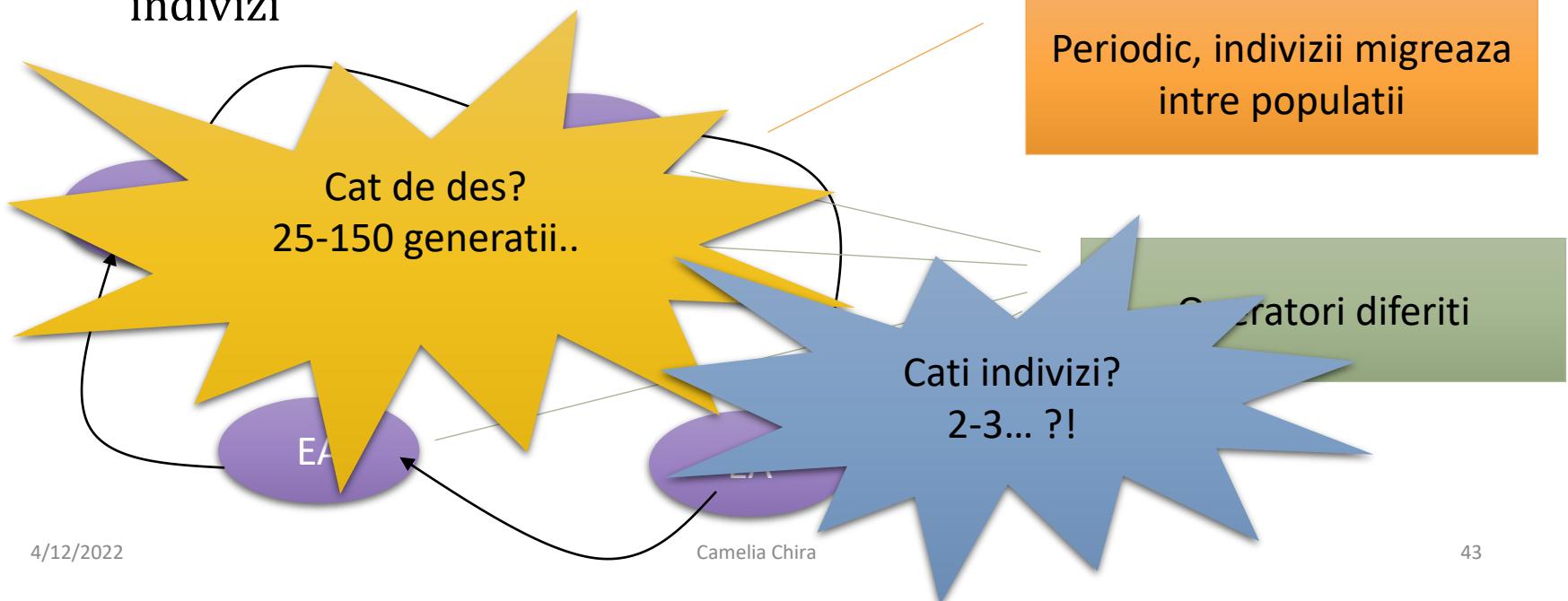
- Prin impunerea unei separari geografice
  - Prin impunerea unui comportament specific - **speciation**
  - Motivatie biologica
    - Specii diferite se adapteaza in natura pentru a ocupa anumite niste ce contin resurse limitate => competitie intre indivizi
    - Reproducerea numai in cadrul unei specii => specii omogene + diferente intre 2 specii
- Recombinare numai cu indivizi similari (nivel fenotipic / genotypic)
- Adauga in reprezentare informatii (initial aleatoare) ce se schimba in functie de recombinare si mutatie; la selectarea unui partener pentru recombinare, alege numai indivizi ce se potrivesc

# Abordari implicate: modele de tip insula

Modele evolutive paralele tip **insula**

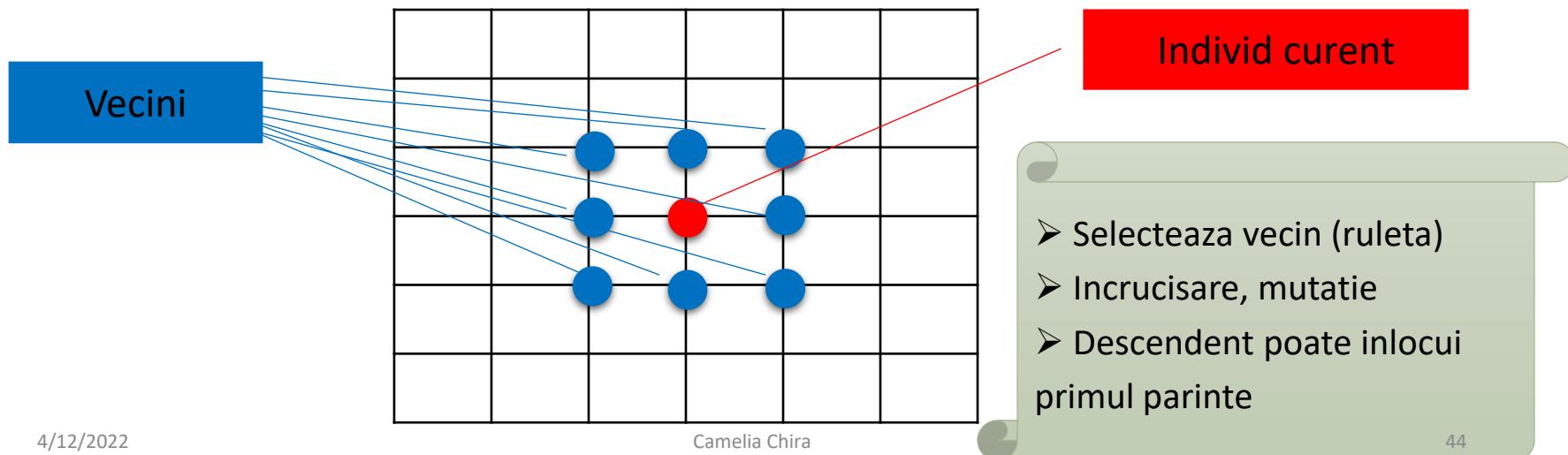
("Island" Model Parallel EAs)

- Populatii multiple evolueaza in paralel (grupate intr-o anumita structura de comunicare – inel, torus)
- Dupa un numar fix de generatii, populatiile vecine interschimba indivizi



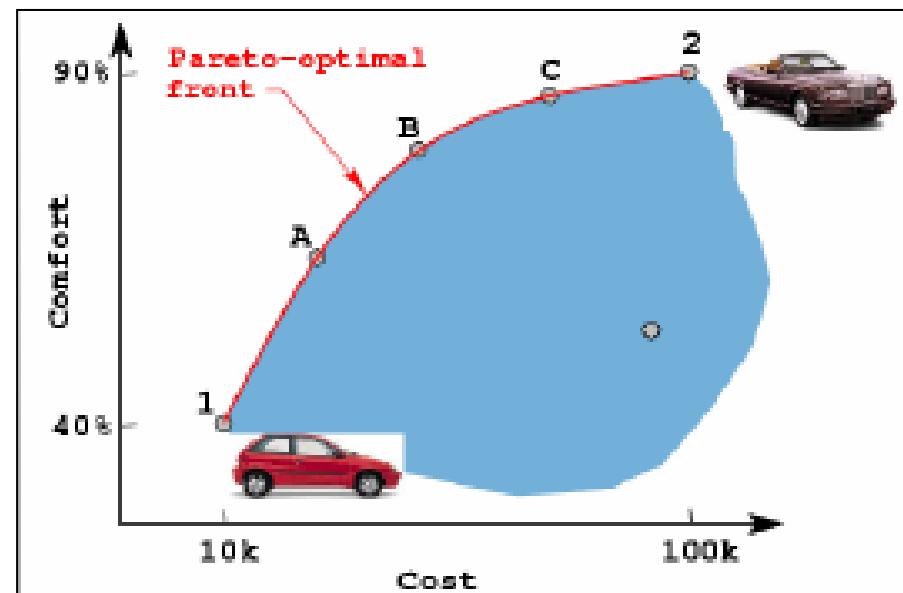
# Abordari implicate: Cellular EA

- Modele paralele **difuze**
- Impunerea unei structuri spatiale populatiei
- Fiecare individ exista intr-un punct din grid
- Selectia pentru recombinare si supravietuire foloseste notiunea de vecinatate => diferite parti ale gridului – cautare in diferite parti ale spatiului (difuziunea solutiilor bune in grid)



# Optimizare multicriteriala

- Sau *vectorială* sau *multiobiectiv*
- Existenta simultana a mai multor **criterii** de optim
- Criteriile pot fi *contradictorii*



# Vectorul criteriu si vectorul solutie

- Optimizare in raport cu  $m$  functii

Vectorul criteriu

Vectorul solutie

$$f_i : R^n \rightarrow R$$

$$F = \begin{pmatrix} f_1 \\ f_2 \\ \dots \\ f_m \end{pmatrix}$$

$$F : \Omega \rightarrow R^m, \Omega \subset R^n$$

$$\begin{pmatrix} x_1 \\ x_2 \\ \dots \\ x_m \end{pmatrix}$$

Cum pot fi  
comparate 2  
solutii?

# Optimizarea Pareto

- Defineste o relatie de dominare intre solutii

Definitie. O solutie **nedominata** este o solutie care:

- Nu este mai slaba decat celelalte solutii
- Este mai buna decat orice alta solutie in raport cu cel putin o functie obiectiv  $f_i$

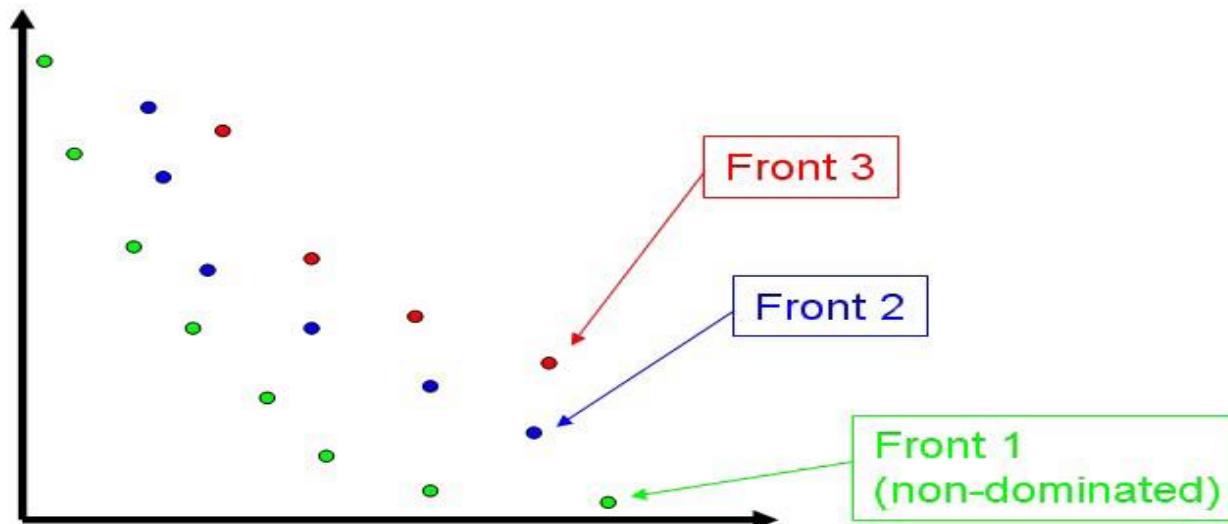
$$\begin{cases} f_i(x) \rightarrow \max \\ i = 1, 2, \dots, m \\ x \in \Omega \end{cases}$$

Multimea valorilor functiei F

$$V = \{ y \in R^m \mid \exists x \in \Omega, y = F(x) \}$$

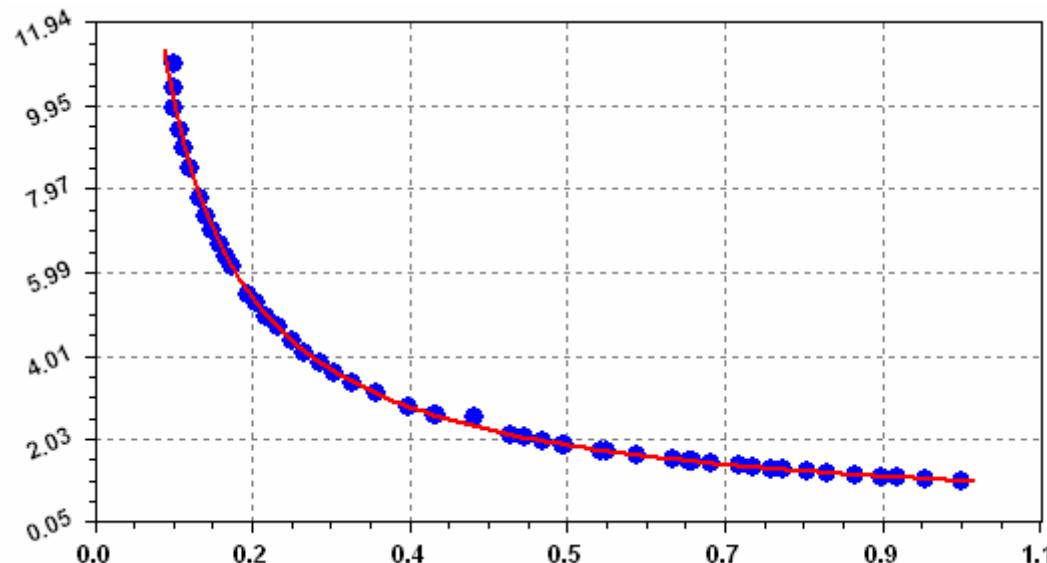
# NSGA

- NSGA: Non-dominated Sorting Genetic Algorithm (NSGA-II)- Deb et al, 2000
  - Rapid, elitist, cu sortare nedominata
  - Selectie turnir, incrusiari, mutatii
- Sortare nedominata a populatiei pe ranguri
  - membrii de rang n domina membrii de rang > n
  - Multime nedominata (rang 1)  $\Leftrightarrow$  front Pareto



# Exemplu

- $f_1(x,y) = x \rightarrow \min$
- $f_2(x,y) = (1+y) / x \rightarrow \min$



# Proiectarea AE



- Reprezentare
- Evaluarea indivizilor: functia de fitness
- Specificarea unor operatori potriviti pentru
  - Mutatie
  - Incrucisare
- Selectia:
  - Selectia indivizilor ce vor fi parinti
  - Selectia indivizilor pentru supravietuire
- Start: Initializare
- Stop: Criteriul de terminare
- Pasii algoritmului: ce se intampla intr-o generatie?

# Experimente

- Nici o concluzie nu poate fi trasa dintr-o singura rulare
  - Numar suficient de rulari independente
  - Masuri statistice: media, deviatia standard
  - Teste statistice
- Comparatii cu alti algoritmi
- Ce masuram?
  - Rezultatul mediu obtinut intr-un anumit timp
  - Timpul mediu necesar obtinerii unui anumit rezultat
  - Proportia de rulari in care s-a obtinut o anumita solutie
  - Cea mai buna solutie din n rulari
- Time units?
  - **Timp necesar:** Depinde de calculator, de implementare,...
  - **Numar generatii:** Daca alti parametri se schimba ex. Marimea populatiei...?

# Evaluarea prin experimente

- Rezolvarea eficienta a unei probleme – solutii bune
- Algoritm propus este mai bun decat alte modele
- Algoritm propus este mai bun decat algoritmii traditionali
- Parametrii algoritmului propus– eficienti?, cum influenteaza performanta?
- Intelegerea comportamentului algoritmului pentru problema aleasa
- Scalabilitate



**BABEŞ-BOLYAI UNIVERSITY**  
Faculty of Mathematics and Computer Science



# Inteligentă Artificială

9: Învățare Automată

Camelia Chira

[cchira@cs.ubbcluj.ro](mailto:cchira@cs.ubbcluj.ro)

# Sisteme inteligente

*"How can we build computer systems that automatically improve with experience, and what are the fundamental laws that govern all learning processes?"*

- Cum sa proiectam roboti mobili autonomi care invata din experienta?
  - Cum sa folosim inregistrari medicale existente pentru a invata ce pacienti viitori vor raspunde mai bine caror tratamente?
  - Cum sa dezvoltam motoare de cautare care se adapteaza automat intereselor individuale ale utilizatorilor?
- 
- *Învățare Automată - Machine Learning (ML)*
    - Dezvoltarea de metode care pot detecta automat forme (patterns) in date si care pot apoi folosi formele descoperite pentru a prezice date viitoare
    - Extragerea de forme si tendinte importante din date pentru a intelege ce ne spun datele

# Exemple ML

- Clasificare
  - Filtru spam
  - Detectarea obiectelor
  - Predictia vremii
- Regresie
  - Predictia vremii
  - Bursa de actiuni
  - Imobiliare
- Ranking
  - Cautare online
  - Gasire elemente similare
- Clustering
  - Gruparea elementelor similare
  - Gruparea genelor
  - Gruparea paginilor web

# Aplicatii

- Recunoaștere de imagini și semnal vocal
- Recunoașterea scrisului de mână
- Detecția fețelor
- Detecția obstacolelor
- Recunoașterea amprentelor
- Controlul roboților
- Predicția vremii
- Diagnosticare medicală
- Detecția fraudelor
- etc

# Învățare Automată: tipuri de învățare

- *Invatare supervizata*
  - Regresie
    - Furnizarea unei iesiri corecte pentru o intrare noua
    - Output continuu – numar real
    - Ex. Predictia preturilor
  - Clasificare
    - Clasificarea unei intrari noi
    - Output discret – eticheta intr-o multime predefinita
    - Ex. Detectarea tumorilor maligne
- *Invatare nesupervizata*
  - Clustering
    - Gasirea unui model sau structuri utile a datelor
    - Output grupuri de date - datele se impart in grupuri astfel incat datele din acelasi grup sa fie similare iar cele din grupuri diferite sa fie foarte diferite
    - Ex. Analiza genelor

# Exemple: probleme de regresie

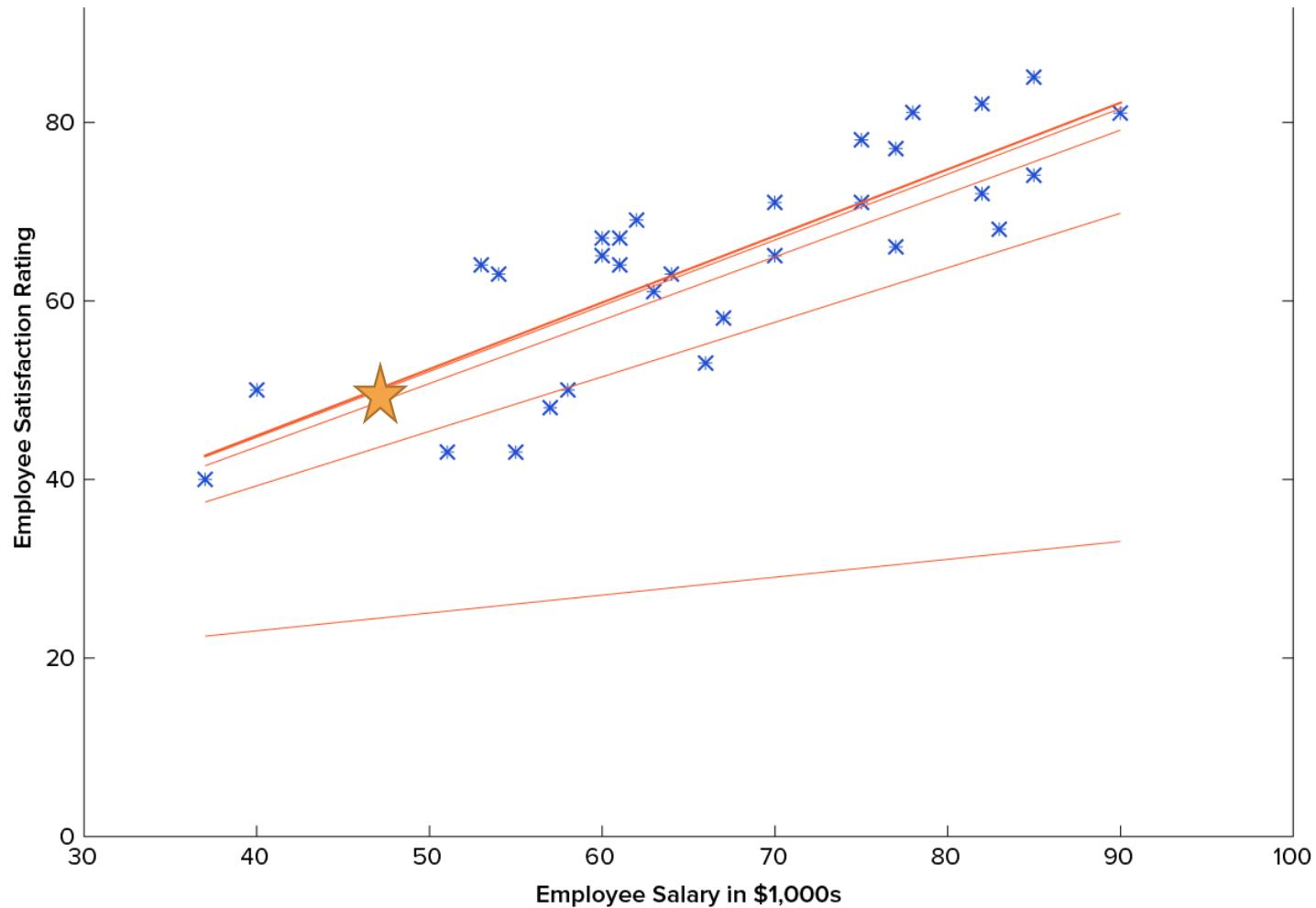
- Predictia pretului unei actiuni pe baza unui istoric
- Predictia nivelului de satisfactie a angajatilor pe baza castigurilor salariale
- Predictia valorii unei case pe baza unor attribute e.g. suprafata, locatie, numar bai, numar dormitoare, gradina, etc

## Exemple modele

- Linear regression
- Logistic regression
- Arbori de decizie
- Retele neuronale



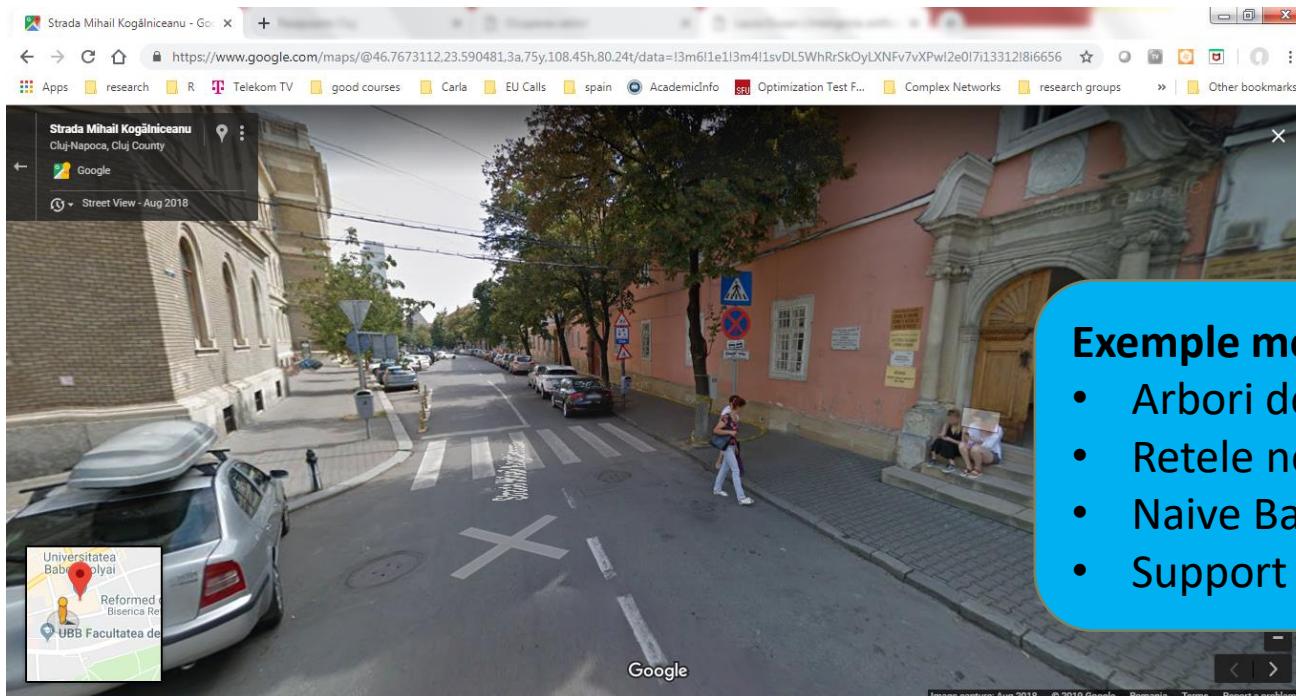
# Exemplu: predictia nivelului de satisfactie



Sursa: <https://www.toptal.com/machine-learning/machine-learning-theory-an-introductory-primer>

# Exemple: probleme de clasificare

- Diagnoza medicală: clasificarea celulelor tumorale în benigne sau maligne
- Spam filtering: clasificarea emailurilor ca spam sau utile
- Face detection: detectarea fetelor în imagini



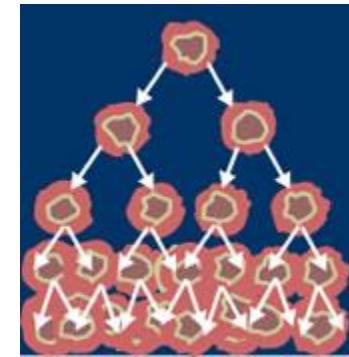
## Exemple modele

- Arbo里 de decizie
- Retele neuronale
- Naive Bayes
- Support Vector Machines

# Exemplu: diagnoza medicală

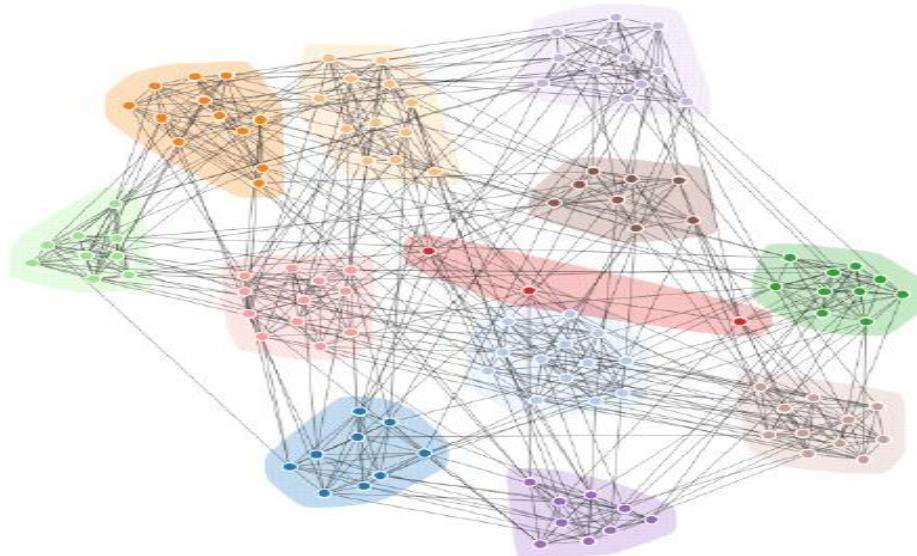
Exemplu de set de date (breast-cancer-wisconsin -arff format)

```
@relation wisconsin-breast-cancer
@attribute Clump_Thickness integer [1,10]
@attribute Cell_Size_Uniformity integer [1,10]
@attribute Cell_Shape_Uniformity integer [1,10]
@attribute Marginal_Adhesion integer [1,10]
@attribute Single_Epi_Cell_Size integer [1,10]
@attribute Bare_Nuclei integer [1,10]
@attribute Bland_Chromatin integer [1,10]
@attribute Normal_Nucleoli integer [1,10]
@attribute Mitoses integer [1,10]
@attribute Class { benign, malignant}
@data
5,1,1,1,2,1,3,1,1,benign
5,4,4,5,7,10,3,2,1,benign
3,1,1,1,2,2,3,1,1,benign
8,10,10,8,7,10,9,7,1,malignant
1,1,1,1,2,10,3,1,1,benign
```



# Exemple: probleme de clustering

- Clasificarea documentelor în funcție de topic (categorii multiple)
- Gruparea clientilor în funcție de interese, istoricul cumpărăturilor sau monitorizarea activităților
- Gruparea genelor din date de tip time-series microarray

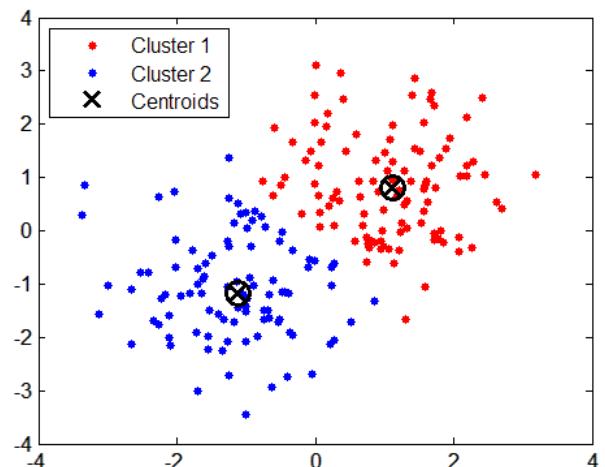


## Exemple modele

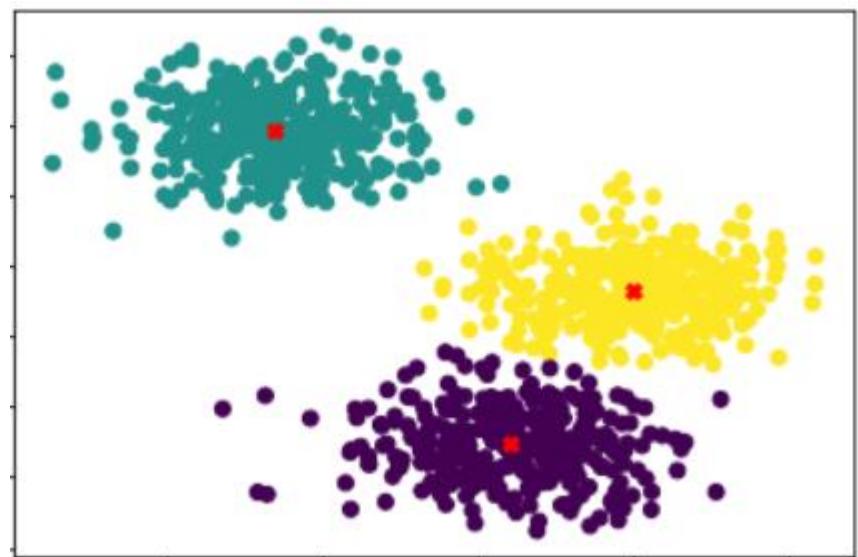
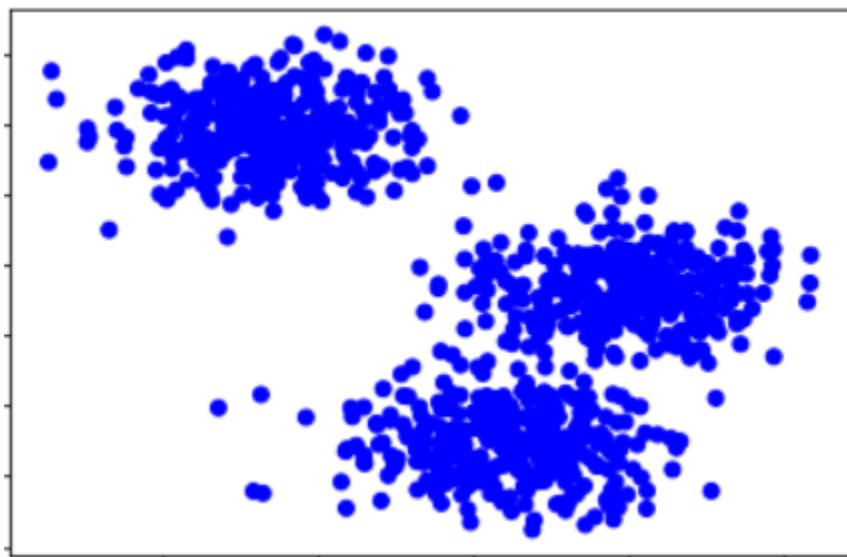
- K-means
- Fuzzy C-means
- Hierarchical clustering

# Exemplu: clustering people

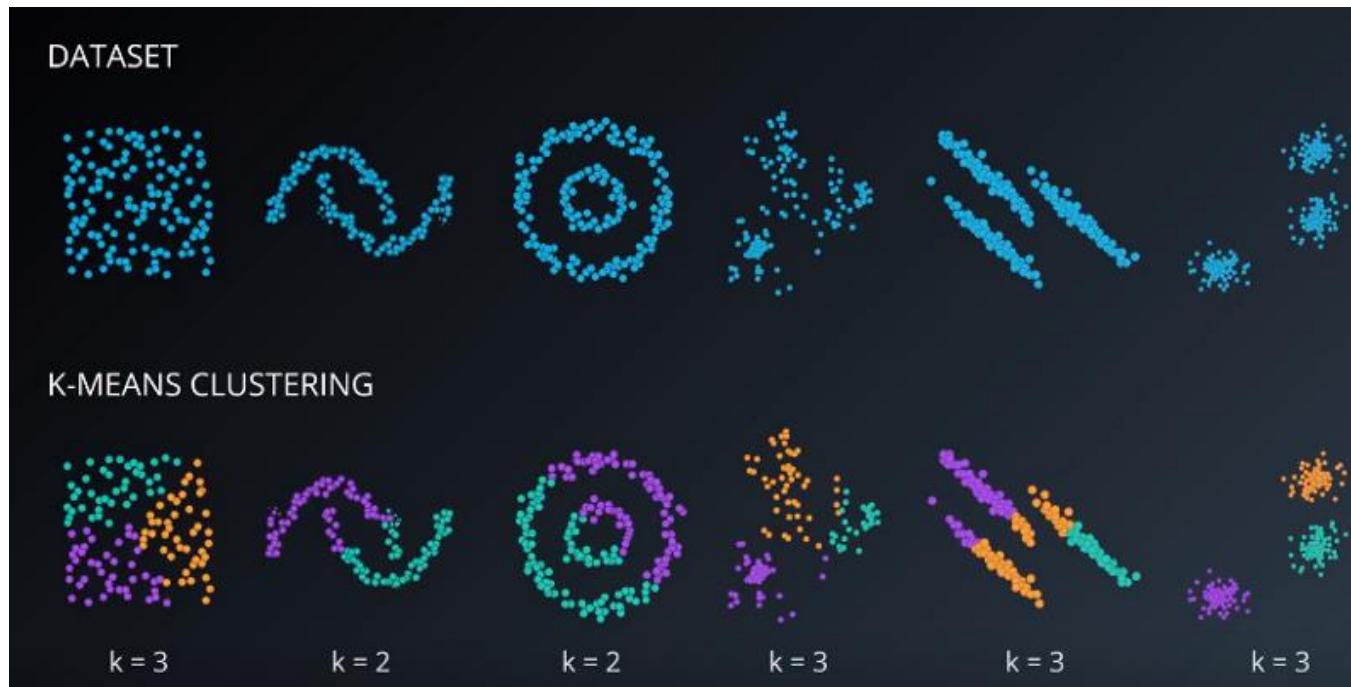
- Identificarea clientilor cu aceleasi interese pe baza cumparaturilor online



## K means



# K means



See <https://scikit-learn.org/stable/modules/clustering.html>

# Concepțe de baza

- Ce se cunoaste?
  - O colecție de înregistrări pentru care se cunoaste clasa careia ii aparțin (set de date etichetate)
  - Fiecare înregistrare conține un set de atribuții și eticheta clasei de care aparține
- Ce se cauta?
  - Un model pentru relația dintre clasa unei înregistrări și atribuții
  - Acest model de clasificare mapează o înregistrare cu anumite atribuții la o anumita clasa
- De ce?
  - Scopul este de a utiliza modelul de învățare gasit pentru a putea fi aplicat unor date noi și determina clasa corespunzătoare pe baza atributelor

# Modele de clasificare

- Proces: etapa de antrenare, etapa de clasificare
- Performanta algoritmului
  - Acuratetea calculata in faza de antrenare, faza de testare
    - $Acc = \text{nr de exemple corect clasificate} / \text{nr total de exemple}$
- Metode de evaluare
  - Seturi disjuncte de antrenare si testare
    - setul de antrenare poate fi impartit in date de invatare si date de validare
    - setul de antrenare este folosit pentru estimarea parametrilor modelului (cei mai buni parametri obtinuti pe validare vor fi folositi pentru constructia modelului final)
  - Validare incruisata cu mai multe ( $h$ ) sub-seturi egale ale datelor (de antrenament)
    - separararea datelor de  $h$  ori in  $h-1$  sub-seturi pentru invatare si 1 sub-set pt validare
    - dimensiunea unui sub-set = dimensiunea setului /  $h$
    - performanta este data de media pe cele  $h$  rulari (ex.  $h = 5$  sau  $h = 10$ )
  - Leave-one-out cross-validation

# Masuri de performanta

- Fie o problema de clasificare cu 2 clase:
  - Clasa C1 - pozitiva
  - Clasa C2 - negativa
- *Matricea de confuzie:*



		Rezultat clasificator	
		C1 (clasa pozitiva)	C2 (clasa negativa)
Clasa reala	C1 (clasa pozitiva)	TP	FN
	C2 (clasa negativa)	FP	TN

**TP: True Positive** = nr de cazuri C1 care sunt clasificate (*corect*) în C1

**TN: True Negative** = nr de cazuri C2 care sunt clasificate (*corect*) în C2

**FP: False Positive** = nr de cazuri C2 dar care sunt clasificate (*incorrect*) în C1

**FN: False Negative** = nr de cazuri C1 dar care sunt clasificate (*incorrect*) în C2

# Masuri de performanta

- **Acuratete =  $(TP+TN)/(TP+TN+FP+FN)$** 
  - nr date clasificate corect / nr total de date
- **Sensitivitate =  $TP/(TP+FN)$** 
  - TP rate sau **recall** (rata de regasire)
- **Specificitate =  $TN/ (TN+FP)$** 
  - TN rate
  - FP rate = 1-specificitate= $FP/(TN+FP)$
- **Precizie =  $TP/(TP+FP)$** 
  - nr cazuri real pozitive / nr cazuri clasificate ca fiind positive (**precision**)

✓ Toate aceste valori sunt in [0,1]  
✓ Valori mari => performanta buna

**TP: True Positive** = nr de cazuri C1 care sunt clasificate (*corect*) în C1

**TN: True Negative** = nr de cazuri C2 care sunt clasificate (*corect*) în C2

**FP: False Positive** = nr de cazuri C2 dar care sunt clasificate (*incorrect*) în C1

**FN: False Negative** = nr de cazuri C1 dar care sunt clasificate (*incorrect*) în C2

# Modele de clasificare

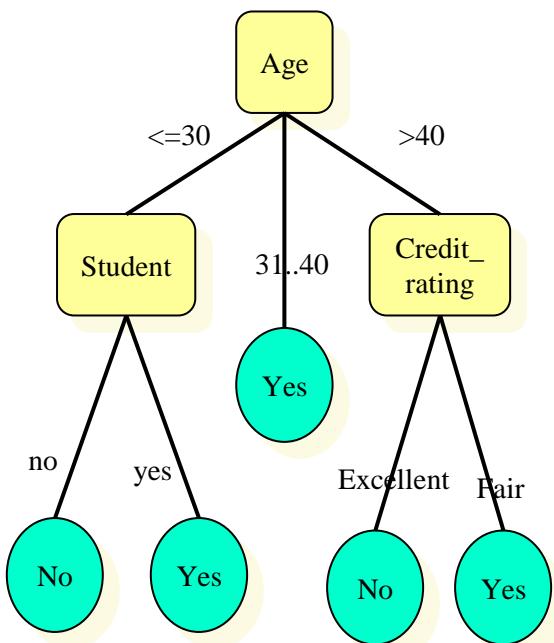
- Un model de clasificare (sau clasificator) trebuie sa fie:
  - Acurat (sa identifice clasa corecta)
  - Compact (usor de interpretat de catre utilizator)
  - Eficient in etapa de antrenare si etapa de clasificare
- Exemple de modele de clasificare
  - Arboi de decizie (decision trees)
  - Reguli de clasificare (classification rules)
  - Retele neuronale (neural networks)
  - Masini cu suport vectorial (Support vector machines)
  - Modele probabiliste
  - etc

# Arbore de decizie

- Scop
  - Divizarea unei colecții de articole în seturi mai mici prin aplicarea succesivă a unor reguli de decizie
- Definire
  - Graf special cu mai multe tipuri de noduri
  - Fiecare nod intern corespunde unui atribut
  - Fiecare ramură de sub un nod (atribut) corespunde unei valori a atributului
  - Fiecare frunză corespunde unei clase
- Tipuri de probleme
  - Exemplele (instanțele) sunt reprezentate printr-un număr fix de atribute, fiecare atribut putând avea un număr limitat de valori
  - Probleme de clasificare: binara / multi-clasa
  - Probleme de regresie

# Arbore de decizie: exemplu

RID	Age	Income	Student	Credit_rating	Class: buys_computer
1	<=30	High	No	Fair	No
2	<=30	High	No	Excellent	No
3	31..40	High	No	Fair	Yes
4	>40	Medium	No	Fair	Yes
5	>40	Low	Yes	Fair	Yes
6	>40	Low	Yes	Excellent	No
7	31..40	Low	Yes	Excellent	Yes
8	<=30	Medium	No	Fair	No
9	<=30	Low	Yes	Fair	Yes
10	>40	Medium	Yes	Fair	Yes
11	<=30	Medium	Yes	Excellent	Yes
12	31..40	Medium	No	Excellent	Yes
13	31..40	High	Yes	Fair	Yes
14	>40	Medium	No	Excellent	No



# Arborei de decizie: proces

- Construirea (creșterea, inducția) arborelui
  - Se bazează pe un set de date de antrenament
  - Lucrează de jos în sus sau de sus înn jos (prin divizare – *splitting*)
- Utilizarea arborelui ca model de rezolvare a problemelor
  - Ansamblul deciziilor efectuate de-a lungul unui drum de la rădăcină la o frunză formează o regulă
  - Regulile formate în arbore sunt folosite pentru etichetarea unor noi date
- Tăierea (curățirea) arborelui (pruning)
  - Se identifică și se mută/elimină ramurile care reflectă zgomote sau excepții

# Construirea arborelui

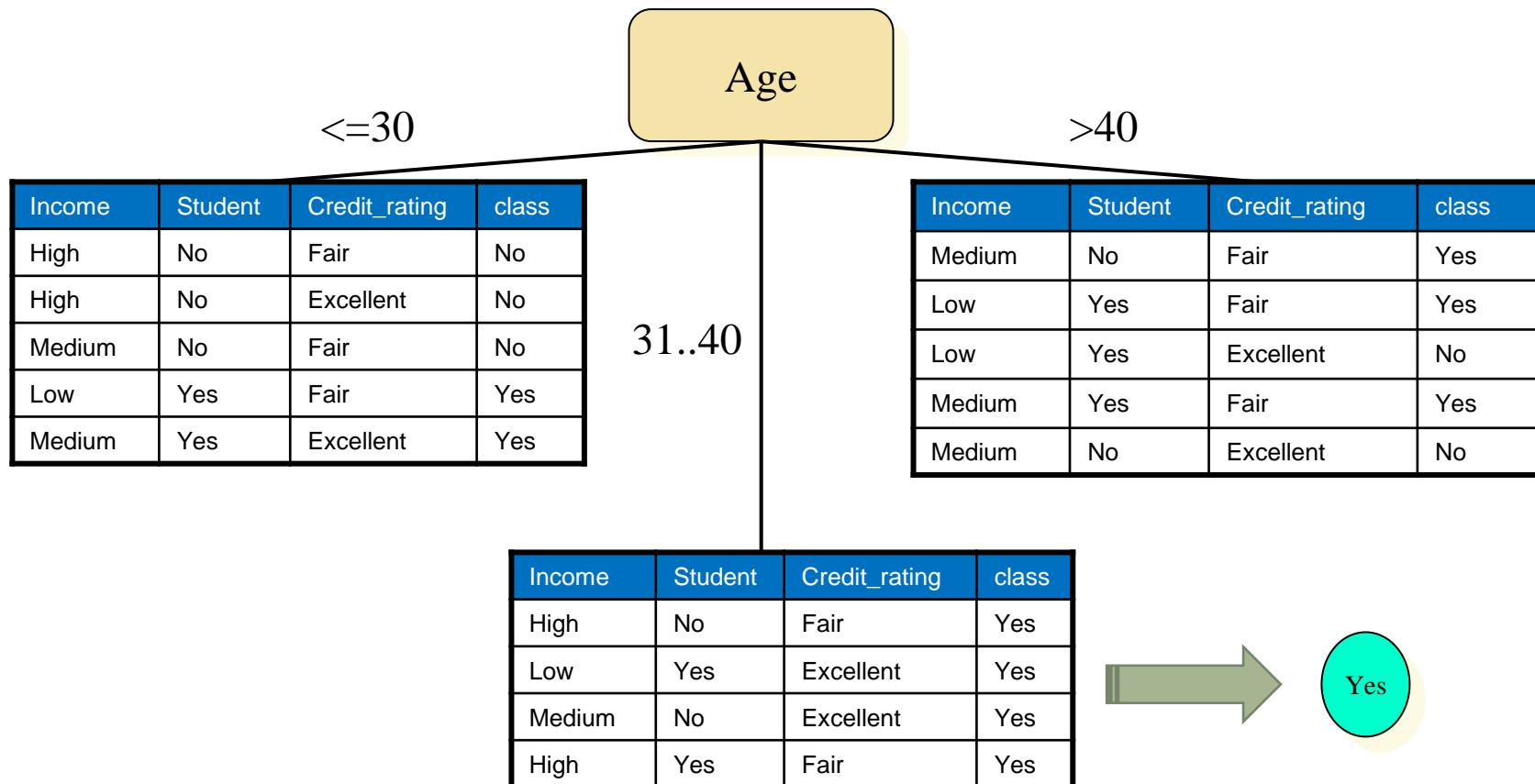
- Divizarea datelor de antrenament în subseturi pe baza caracteristicilor datelor
  - Un nod - întrebare legată de o anumită proprietate a unui obiect dat
  - Ramurile ce pleacă din nod - etichetate cu posibilele răspunsuri la întrebarea din nodul curent
  - La început toate exemplele sunt plasate în rădăcină (la pornire, un atribut va fi rădăcina arborelui, iar valorile atributului vor deveni ramuri ale rădăcinii)
  - Pe următoarele nivele, exemplele sunt partaționate în funcție de atrbute - ordinea considerării atributelor (pentru fiecare nod se alege în mod recursiv câte un atribut - cu valorile lui pe ramurile descendente din nodul curent)
- *Proces iterativ: reguli de oprire*
  - Toate exemplele aferente unui nod fac parte din aceeași clasă - nodul devine frunză și este etichetat cu  $G_i$
  - Nu mai sunt exemple - nodul devine frunză și este etichetat cu clasa majoritară în setul de date de antrenament
  - Nu mai pot fi considerate noi atrbute

# Construirea arborelui: exemplu

RID	Age	Income	Student	Credit_rating	Class: buys_computer
1	<=30	High	No	Fair	No
2	<=30	High	No	Excellent	No
3	31..40	High	No	Fair	Yes
4	>40	Medium	No	Fair	Yes
5	>40	Low	Yes	Fair	Yes
6	>40	Low	Yes	Excellent	No
7	31..40	Low	Yes	Excellent	Yes
8	<=30	Medium	No	Fair	No
9	<=30	Low	Yes	Fair	Yes
10	>40	Medium	Yes	Fair	Yes
11	<=30	Medium	Yes	Excellent	Yes
12	31..40	Medium	No	Excellent	Yes
13	31..40	High	Yes	Fair	Yes
14	>40	Medium	No	Excellent	No

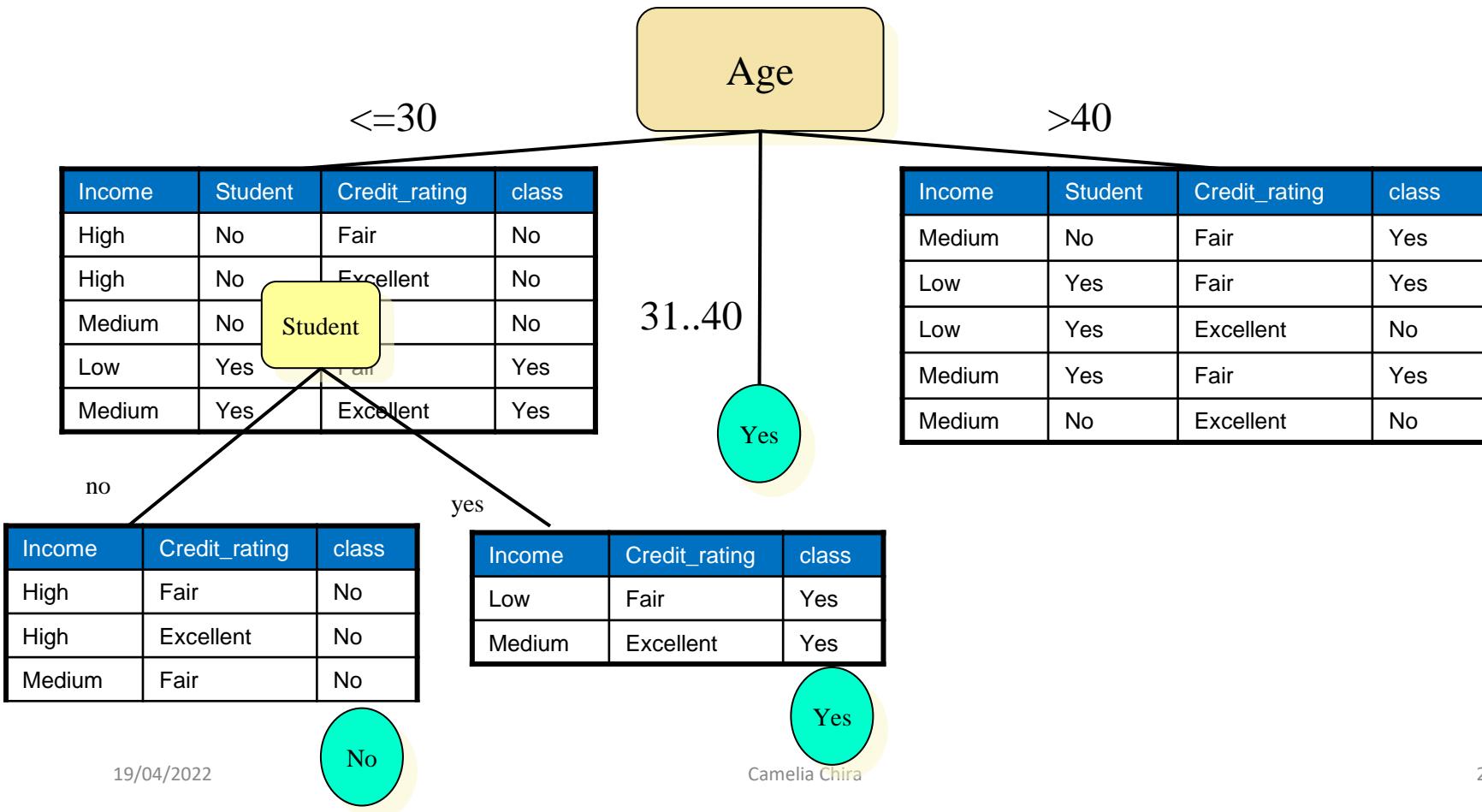
# Construirea arborelui: exemplu

- Pentru radacina se alege atributul 'Age'



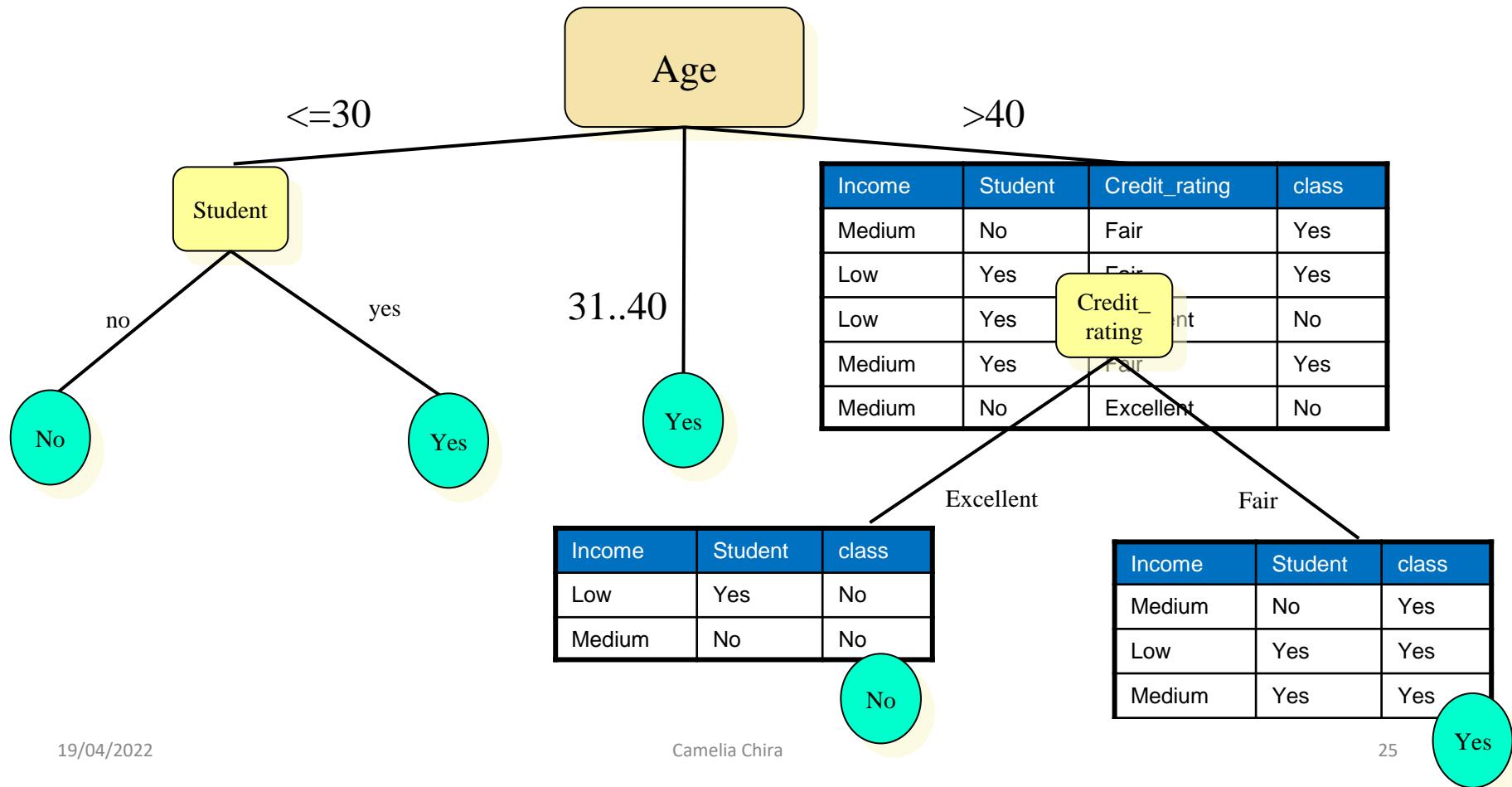
# Construirea arborelui: exemplu

- Pentru ramura  $age \leq 30$  se alege atributul ‘*Student*’

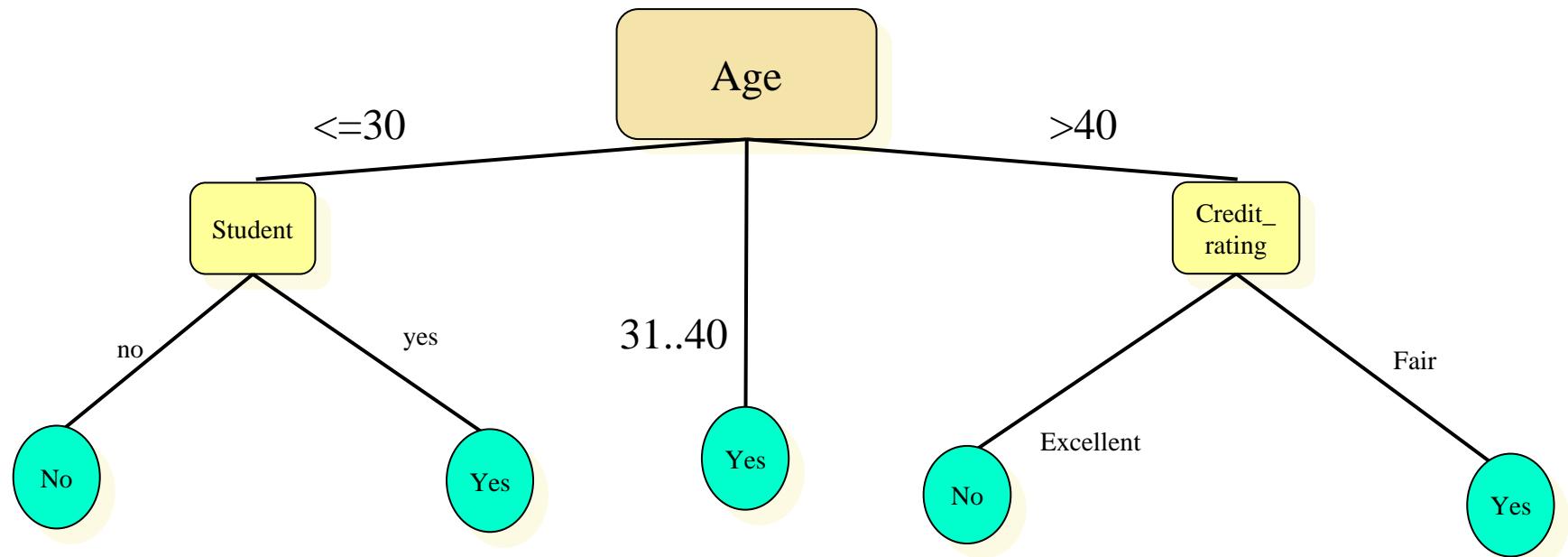


# Construirea arborelui: exemplu

- Pentru ramura  $\text{age} > 40$  se alege atributul '*Credit\_rating*'



# Construirea arborelui: exemplu



# Algoritmi pentru construirea arborelui de decizie

- Algoritmul ID3
  - Intrare: set de date
  - Iesire: arbore de decizie (noduri interne etichetate cu atrbute, noduri frunză etichetate cu clase, muchii etichetate cu valori ale atributelor)
- Algoritmul C4.5
  - Imbunatatire a algoritmul ID3 pentru a trata atrbute continue, valori absente
- J48 –implementarea din Weka a algoritmului C4.5
  - <https://www.cs.waikato.ac.nz/ml/weka/>

# Algoritmul ID3/C4.5

**generare(D, A){** //D – partitōnare a exemplelor de antrenament, A – lista de atrbute

Crearea unui nod nou N

**Dacă** exemplele din D fac parte dintr-o singură clasă C **atunci**

nodul N devine frunză și este etichetat cu C

returnează nodul N

**Altfel**

**Dacă**  $A = \emptyset$  **atunci**

nodul N devine frunză și este etichetat cu clasa majoritară în D

**returnează** nodul N

**Altfel**

atribut\_separare = *Selectează\_atribut*(D, A)

Etichetează nodul N cu atribut\_separare

**Pentru** fiecare valoare posibilă vj a lui atribut\_separare

Fie Dj mulțimea exemplelor din D pentru care atribut\_separare = vj

**Dacă**  $D_j = \emptyset$  **atunci**

Atașează nodului N o frunză etichetată cu clasa majoritară în D

**Altfel**

Atașează nodului N un nod returnat de generare( $D_j, A - \text{atribut\_separare}$ )

**returnează** nodul N

}

# Algoritm ID3/C4.5: alegerea atributului de ramificare

- *Selectează\_atribut(D, A)*
  - Aleatoare
  - Atributul cu cele mai puține/multe valori
  - Atributul cu cel mai mare câștig de informație (information gain)
    - Rata câștigului (gain ratio)
- Câștigul de informație
  - O măsură de impuritate
    - 0 (minimă) dacă toate exemplele fac parte din aceeași clasă
    - 1 (maximă) dacă avem număr egal de exemple din fiecare clasă
  - Se bazează pe entropia datelor
    - Clasificare binară:  $H(D) = - p_1 \log_2 p_1 - p_2 \log_2 p_2$ ,  $p_1$  proporția exemplelor pozitive în setul de date D,  $p_2$  proporția exemplelor negative în setul de date D
    - Clasificare cu mai multe clase:  $H(D) = \sum_i - p_i \log_2 p_i$
  - Câștigul de informație al unui atribut al datelor
    - Reducerea entropiei setului de date ca urmare a eliminării atributului

# Alegerea atributului de ramificare - Information Gain

- Set de date distribuit in k clase:  $D=\{C_1, C_2, \dots, C_k\}$
- Distribuția de probabilitate  $(p_1, p_2, \dots, p_k)$ ,  $p_i = \text{card}(C_i)/\text{card}(D)$
- Fie A un atribut și  $v_1, v_2, \dots, v_{m_A}$  valorile posibile ale acestui atribut
- Fie  $D_j$  setul de date din D pt care atributul A are valoarea  $v_j$  și  $P_j$  distribuția datelor din  $D_j$  în cele k clase
- $C_{ji} =$  datele din clasa  $C_i$  care au valoarea  $v_j$  pt atributul A
- **Câștigul informational - Information Gain (IG)** obtinut prin partitionarea setului de date folosind atributul A:

$$IG(D, A) = H(D) - \sum_{j=1}^{m_A} P(D_j | A = v_j) H(D_j | A = v_j)$$

$$H(D) = - \sum_{i=1}^k p_i \log p_i \quad P(D_j | A = v_j) = \frac{\text{card}(D_j)}{\text{card}(D)}$$

$$H(D_j | A = v_j) = - \sum_{i=1}^k p_{ij} \log p_{ij}, \quad p_{ij} = \frac{\text{card}(C_{ji})}{\text{card}(C_i)}$$

# Information Gain (IG): Exemplu

RID	Age	Income	Student	Credit_rating	Class: buys_computer
1	<=30	High	No	Fair	No
2	<=30	High	No	Excellent	No
3	31..40	High	No	Fair	Yes
4	>40	Medium	No	Fair	Yes
5	>40	Low	Yes	Fair	Yes
6	>40	Low	Yes	Excellent	No
7	31..40	Low	Yes	Excellent	Yes
8	<=30	Medium	No	Fair	No
9	<=30	Low	Yes	Fair	Yes
10	>40	Medium	Yes	Fair	Yes
11	<=30	Medium	Yes	Excellent	Yes
12	31..40	Medium	No	Excellent	Yes
13	31..40	High	Yes	Fair	Yes
14	>40	Medium	No	Excellent	No

Distribuția claselor

- $C_1 = \text{"yes"}, C_2 = \text{"no"}$
- $p_1 = 9/14, p_2 = 5/14$

$$H(p_1, p_2) = -\frac{9}{14} \log_2 \frac{9}{14} - \frac{5}{14} \log_2 \frac{5}{14}$$

$$H(p_1, p_2) = 0.940$$

# Information Gain (IG): Exemplu

- Calcul entropie pt fiecare atribut

RID	Age	Income	Student	Credit_rating	Class: buys_computer
1	<=30	High	No	Fair	No
2	<=30	High	No	Excellent	No
3	31..40	High	No	Fair	Yes
4	>40	Medium	No	Fair	Yes
5	>40	Low	Yes	Fair	Yes
6	>40	Low	Yes	Excellent	No
7	31..40	Low	Yes	Excellent	Yes
8	<=30	Medium	No	Fair	No
9	<=30	Low	Yes	Fair	Yes
10	>40	Medium	Yes	Fair	Yes
11	<=30	Medium	Yes	Excellent	Yes
12	31..40	Medium	No	Excellent	Yes
13	31..40	High	Yes	Fair	Yes
14	>40	Medium	No	Excellent	No

Age:

$\text{“}<=30\text{”}$ : 3 No, 2 Yes, 5 total  
 $\text{“}31..40\text{”}$ : 0 No, 4 Yes, 4 total  
 $\text{“}>40\text{”}$ : 3 Yes, 2 No, 5 total

$$H(\text{“}<=30\text{”})$$

$$\begin{aligned}
 &= -3/5 \log(3/5) - 2/5 \log(2/5) \\
 &= 0.971
 \end{aligned}$$

$$H(\text{“}31..40\text{”})$$

$$\begin{aligned}
 &= -0 \log(0) - 4/4 \log(4/4) \\
 &= 0
 \end{aligned}$$

$$H(\text{“}>40\text{”})$$

$$\begin{aligned}
 &= -3/5 \log(3/5) - 2/5 \log(2/5) \\
 &= 0.971
 \end{aligned}$$

# Information Gain (IG): Exemplu

- Calcul entropie pt fiecare atribut: *Age*

$$H(\text{age}) = 5/14 * H(" \leq 30 ") + 4/14 * H(" 31..40 ") + 5/14 * H(" > 40 ") = 0.694$$

- *Castigul informational pentru atributul Age:*

$$IG(\text{Age}) = H(p1, p2) - H(\text{age}) = 0.246$$

- Similar, se calculeaza:

$$IG(\text{Income}) = 0.029$$

$$IG(\text{Student}) = 0.151$$

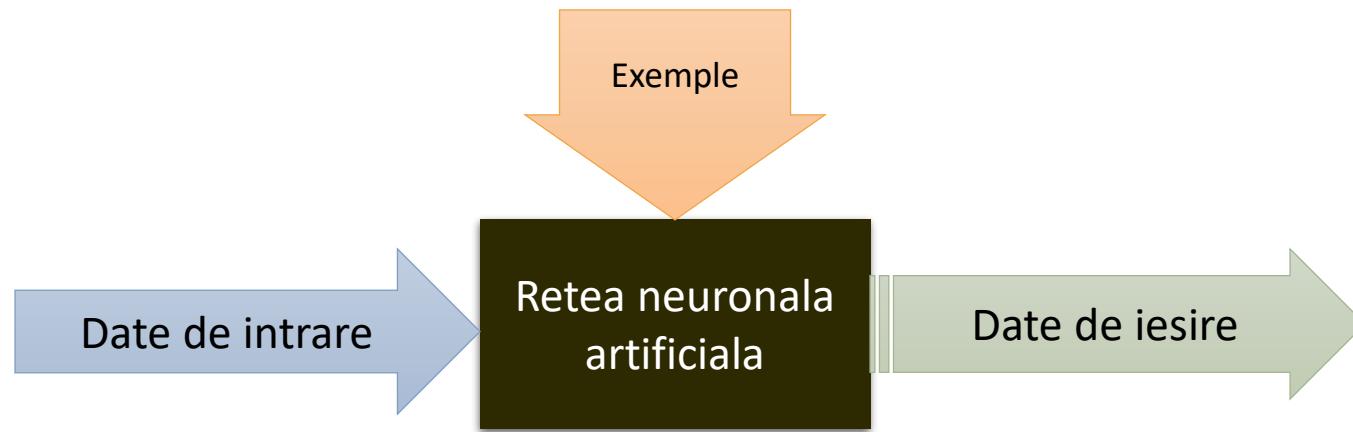
$$IG(\text{Credit\_rating}) = 0.048$$

# Utilizarea arborilor de decizie

- Se extrag regulile formate în arborele anterior construit
- Reguli extrase din arborele dat în exemplul anterior:
  - IF  $age = “<=30”$  AND  $student = “no”$  THEN  $buys\_computer = “no”$
  - IF  $age = “<=30”$  AND  $student = “yes”$  THEN  $buys\_computer = “yes”$
  - IF  $age = “31...40”$  THEN  $buys\_computer = “yes”$
  - IF  $age = “>40”$  AND  $credit\_rating = “excellent”$  THEN  $buys\_computer = “no”$
  - IF  $age = “>40”$  AND  $credit\_rating = “fair”$  THEN  $buys\_computer = “yes”$
- Regulile sunt folosite pentru a clasifica datele de test (date noi). Fie  $x$  o dată pentru care nu se știe clasa de apartenență
  - Regulile se pot scrie sub forma unor predicate astfel:
    - IF  $age(x, <=30)$  AND  $student(x, no)$  THEN  $buys\_computer(x, no)$
    - IF  $age(x, <=30)$  AND  $student(x, yes)$  THEN  $buys\_computer(x, yes)$

# Retele neuronale artificiale

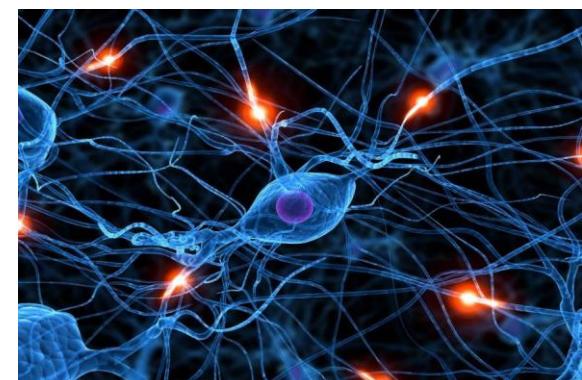
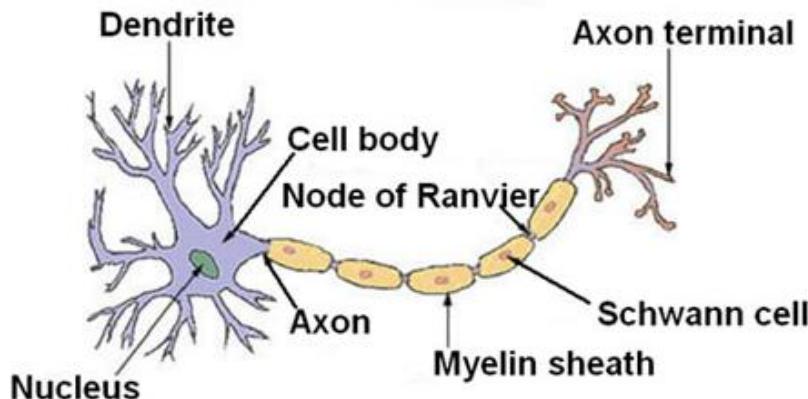
- Sisteme artificiale inspirate de structura si functionarea creierului
- Constituite din mai multe *unitati functionale interconectate (neuroni)*
- **Sistem adaptiv** capabil sa furnizeze raspunsuri pentru o problema dupa ce a fost antrenat pentru probleme similare



- Clasificatori de tip black-box

# Retele neuronale – modelul biologic

- Creierul uman: ~10.000.000.000 de neuroni conectați prin sinapse
- **Neuron**: are un corp (soma), un axon și multe dendrite
- Un neuron poate fi într-una din 2 stări: *activ* – dacă informația care intră în neuron depășește un anumit prag de stimulare, *pasiv* – altfel
- **Sinapsă**: legătura între axon-ul unui neuron și dendritele altui neuron
- 5.000 de conexiuni / neuron (în medie)

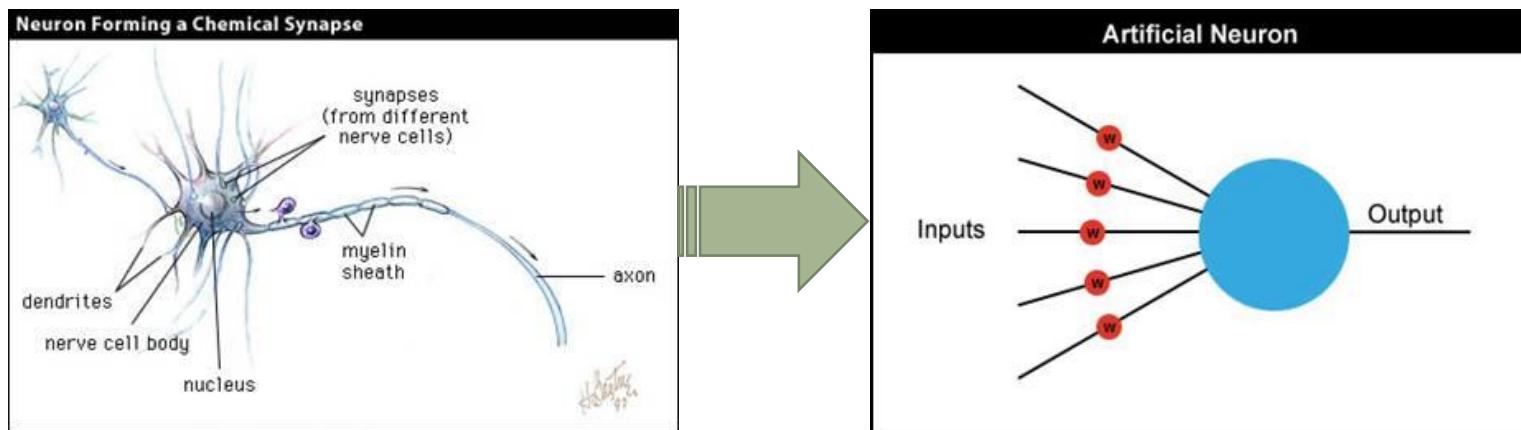


# Retele neuronale

**Retea neuronală:** ansamblu de unitati functionale (neuroni) interconectate

**Unitate functională:** model simplificat al neuronului biologic care efectueaza prelucrari simple asupra unor date de intrare

=> *Model foarte simplificat la creierului*

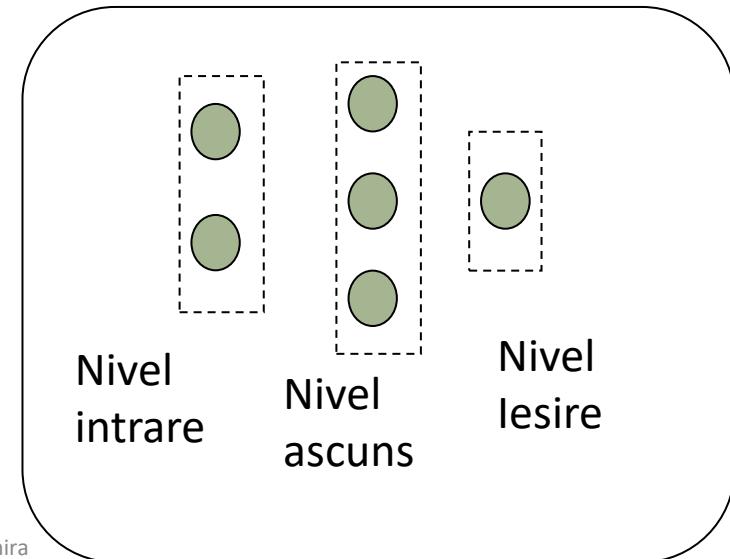


# Retele neuronale artificiale (RNA)

- RNA: set de neuroni artificiali (unități functionale) interconectați
  - Fiecare neuron primește mai multe semnale de intrare și produce un semnal de ieșire
  - Reteaua primește un vector de intrare (prin neuronii de intrare) și produce un vector de ieșire (prin neuronii de ieșire)
- **Arhitectura** = graf orientat etichetat; fiecare arc are asociată o pondere numerică care modelează permeabilitatea sinaptică
- **Funcționare** = procesul prin care RNA transformă un vector de intrare într-un vector de ieșire
- **Antrenare** = procesul prin care sunt stabilite valorile ponderilor sinaptice și ale altor parametri ai rețelei

# Arhitectura

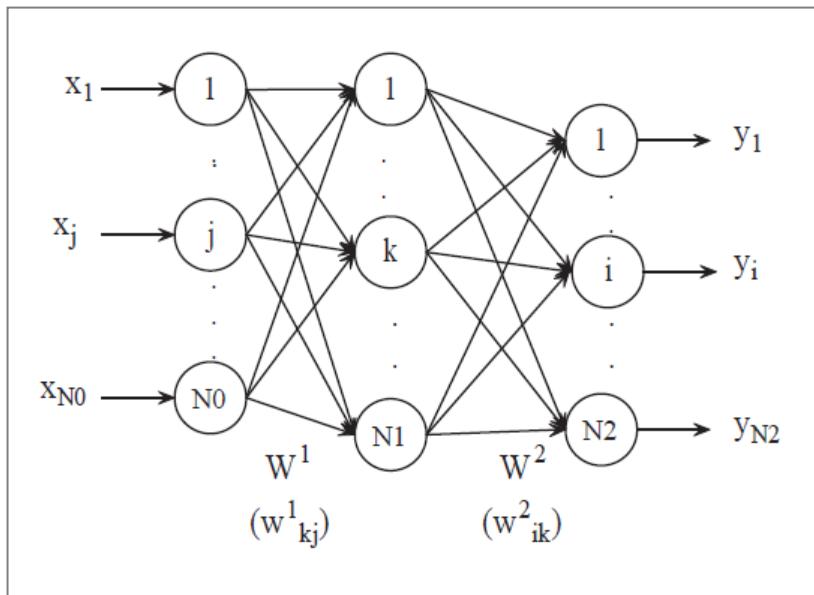
- Modul in care sunt amplasate si interconectate unitatile functionale (neuronii)
- RNA  $\Leftrightarrow$  graf orientat etichetat
  - Noduri: neuroni
  - Arce: modul in care sunt conectati neuronii
  - Etichete: ponderile conexiunilor
- Retele organizate pe nivele
  - Un nivel de unitati de intrare
  - Unul sau mai multe nivele de unitati ascunse
  - Un nivel de iesire



# Principalele tipuri de arhitecturi

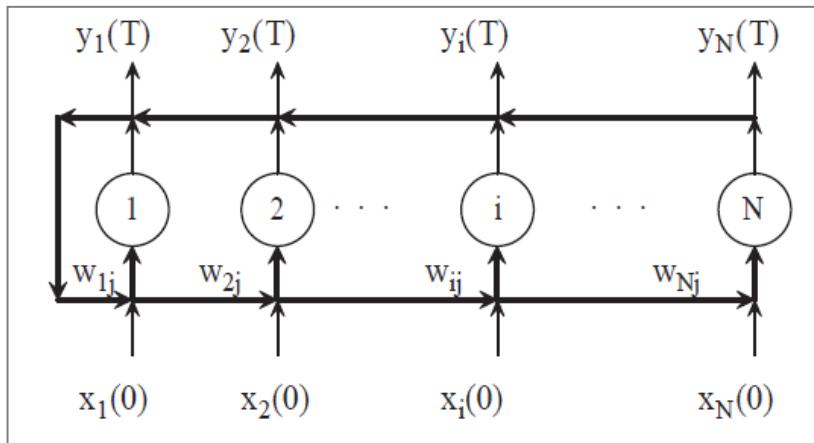
- Feed-forward

- Graful suport nu conține cicluri (neuronii sunt de obicei plasați pe mai multe nivele)
- Semnalul de ieșire poate fi calculat prin compunerea unor funcții de agregare și de activare



- Recurentă

- Graful suport conține cicluri
- Semnalul de ieșire este calculat prin simularea unui sistem dinamic (proces iterativ)

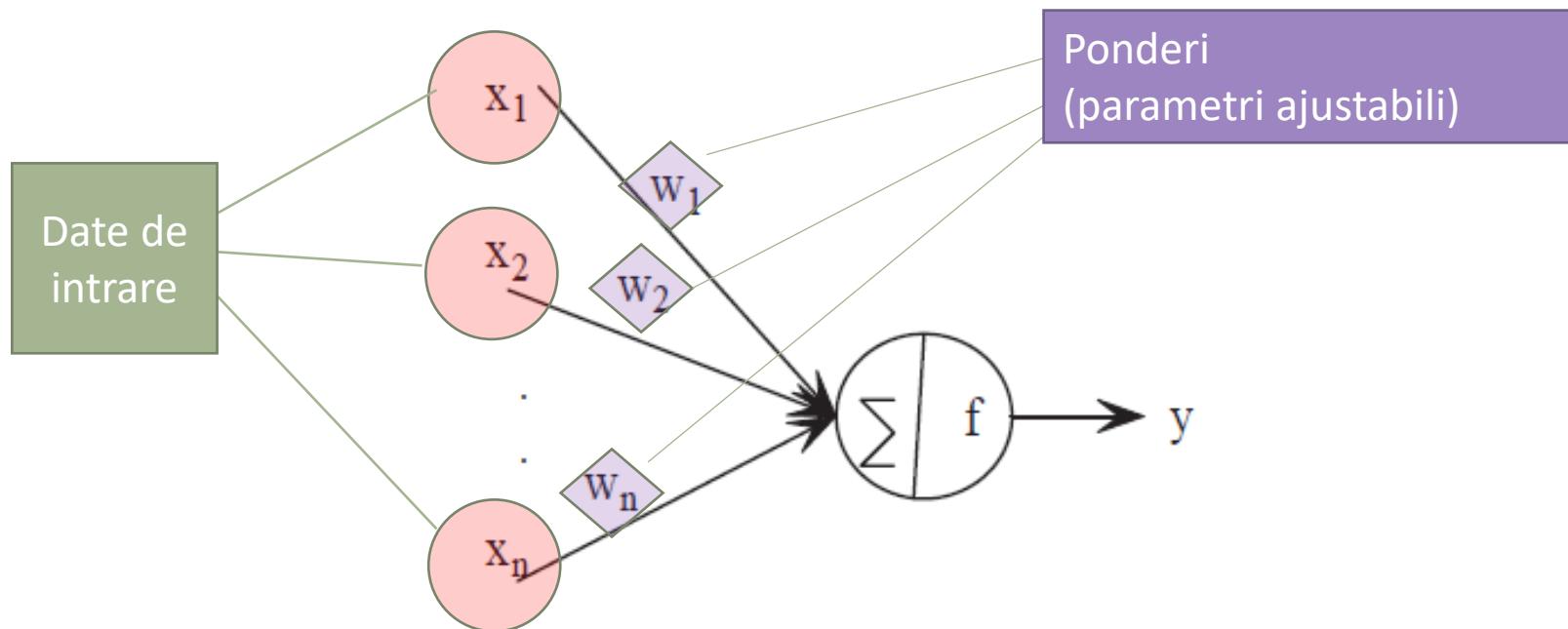


# Proiectarea RNA

- **Alegerea arhitecturii:** număr de nivele, număr de unități pe fiecare nivel, funcții de activare, tip interconectare
- **Antrenare:** determinarea valorilor ponderilor folosind un set de antrenare și un algoritm de învățare
- **Validare/testare:** analiza comportamentului rețelei pentru exemple care nu fac parte din setul de antrenare
- Pentru o problema de clasificare a unor date n-dimensionale in m clase rețeaua ar trebui să aibă:
  - n unități de intrare
  - m unități de ieșire

# Unitati functionale

- Sau *neuroni* sau *elemente de procesare*
- Primeste semnale de intrare si produce un semnal de iesire (forma numérica)



# Rolul unitatilor functionale

- Unitati de intrare
  - Primesc semnale din partea mediului
  - Rol: retransmit semnalul primit catre alti neuroni
- Unitati ascunse
  - Conectate doar cu alte unitati ale retelei
  - Nu comunica direct cu mediul extern
  - Rol: colecteaza si prelucreaza , distribuie semnalul de iesire catre alti neuroni
- Unitati de iesire
  - Rol: colecteaza semnale de la alti neuroni, le prelucreaza si transmit iesirea mediului extern

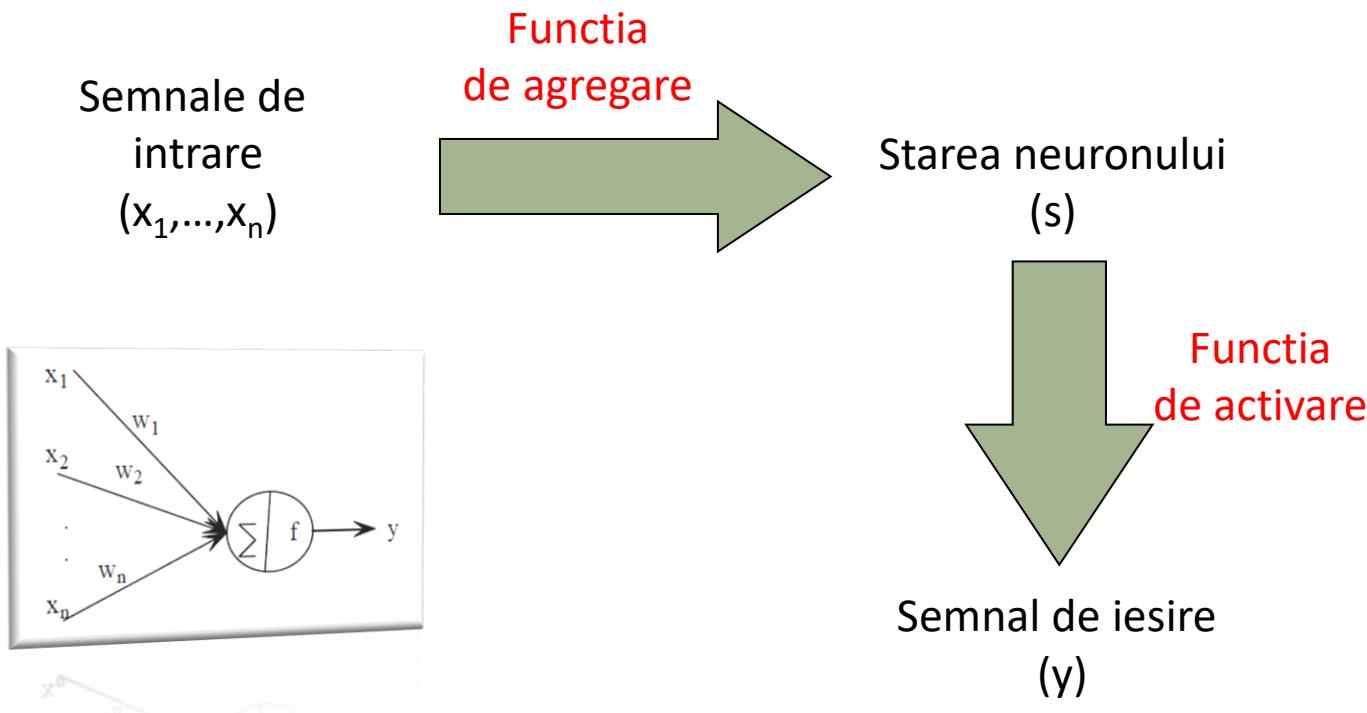
# Model RNA

- Retea cu  $p$  neuroni
- Conectati printr-o multime de *ponderi sinaptice*
- Fiecare neuron are  $n$  intrari si o iesire  $y$
- Intrari:  $x_1, x_2, \dots, x_n$  (numere reale)
- Ponderi sinaptice:  $w_1, w_2, \dots, w_n$  (numere reale)
- Fiecare neuron calculeaza starea sa interna  $s$ :

$$s = \sum_{j=1}^n w_j x_j$$

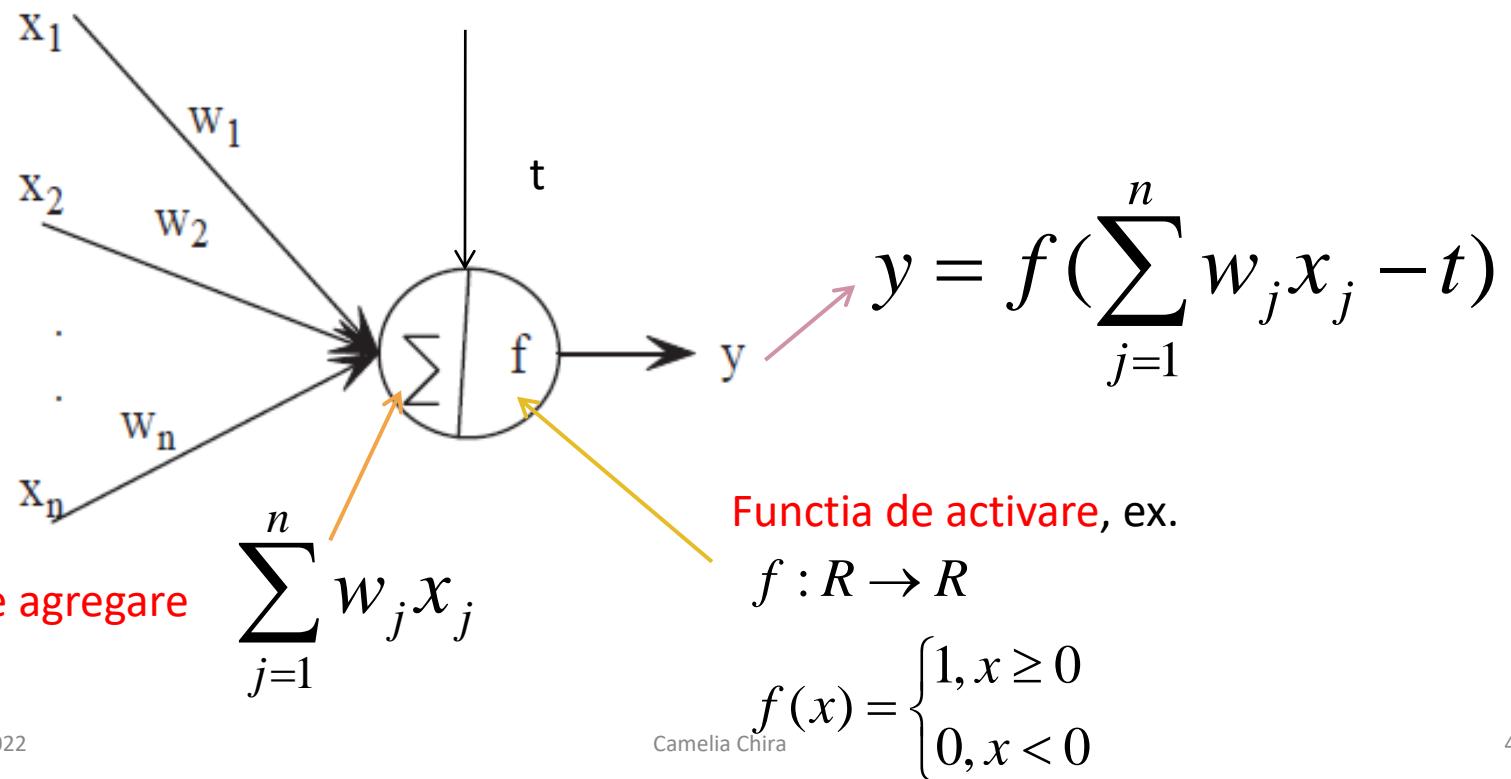
# RNA: un neuron

- Combinarea semnalelor de intrare se realizează printr-o funcție de agregare
- Semnalul de ieșire aplicand o funcție de activare

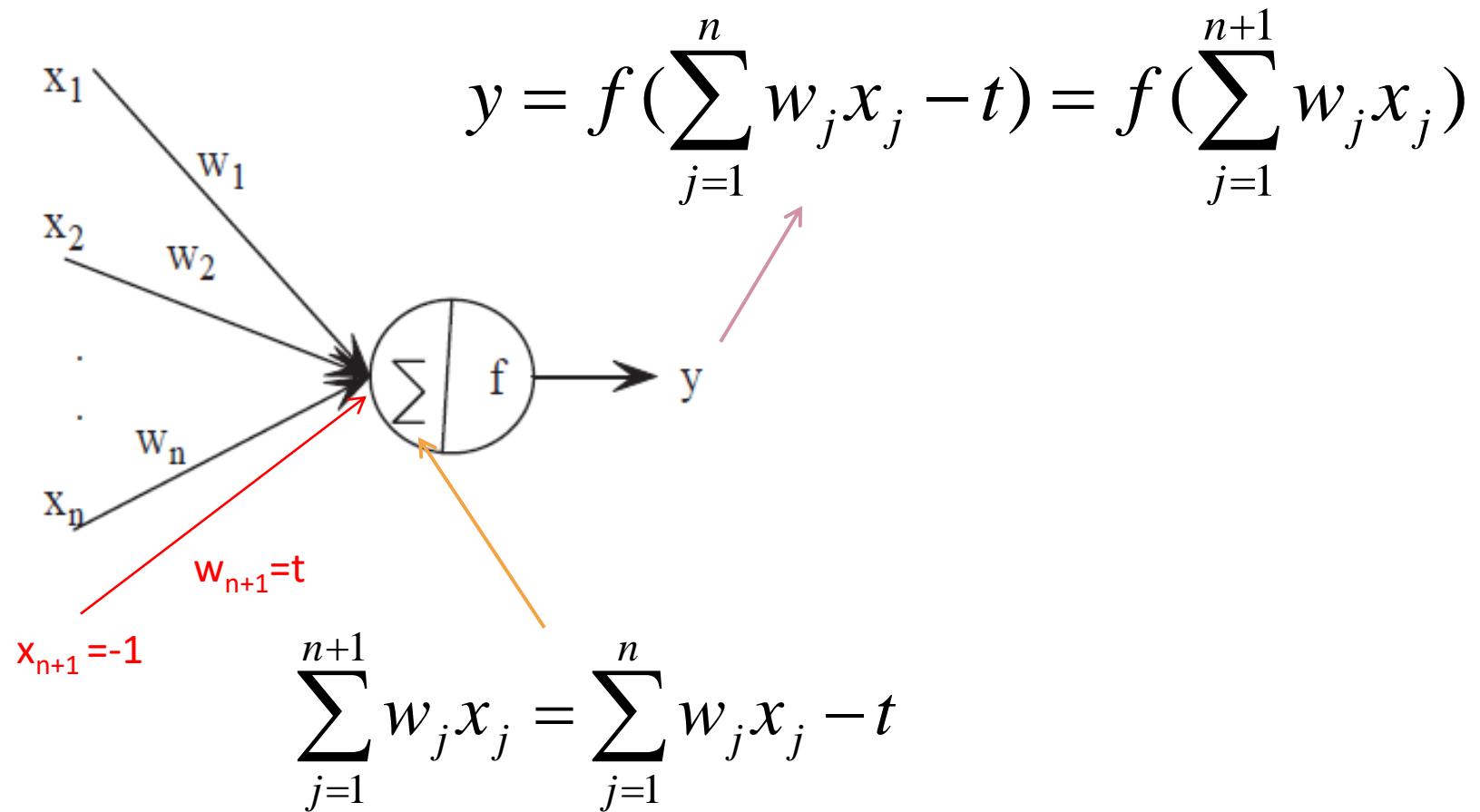


# RNA: un neuron

- Fiecare neuron este caracterizat de un **prag de activare** notat cu  $t$
- Iesirea  $y$  a neuronului este  $+1$  daca activarea totală  $\geq t$  (modelul McCulloch-Pitts)



# RNA: un neurone



# Functia de agregare

Suma ponderata

$$s = \sum_{i=1}^n w_i x_i - t$$

Distanta euclidiana

$$s = \sum_{i=1}^n (w_i - x_i)^2$$

Neuron multiplicativ

$$s = \prod_{i=1}^n x_i^{w_i}$$

Conexiuni de ordin superior

$$s = \sum_{i=1}^n w_i x_i + \sum_{i,j=1}^n w_{ij} x_i x_j + \dots$$

# Functia de activare

- Depinde de modelul de retea neuronala studiat
- Se mai numeste: functie de *raspuns*, functie *neuronala*, functie de *iesire*, functie de *transfer*

Functia prag (Heaviside)

$$f : R \rightarrow \{0,1\}$$

$$f(x) = H(x) = \begin{cases} 1, & x \geq 0 \\ 0, & x < 0 \end{cases}$$

Functie liniara

$$f(x) = x$$

$$f(x) = \max(0, x)$$

Functia signum

$$f : R \rightarrow \{-1,1\}$$

$$f(x) = \text{sgn}(x) = \begin{cases} 1, & x \geq 0 \\ -1, & x < 0 \end{cases}$$

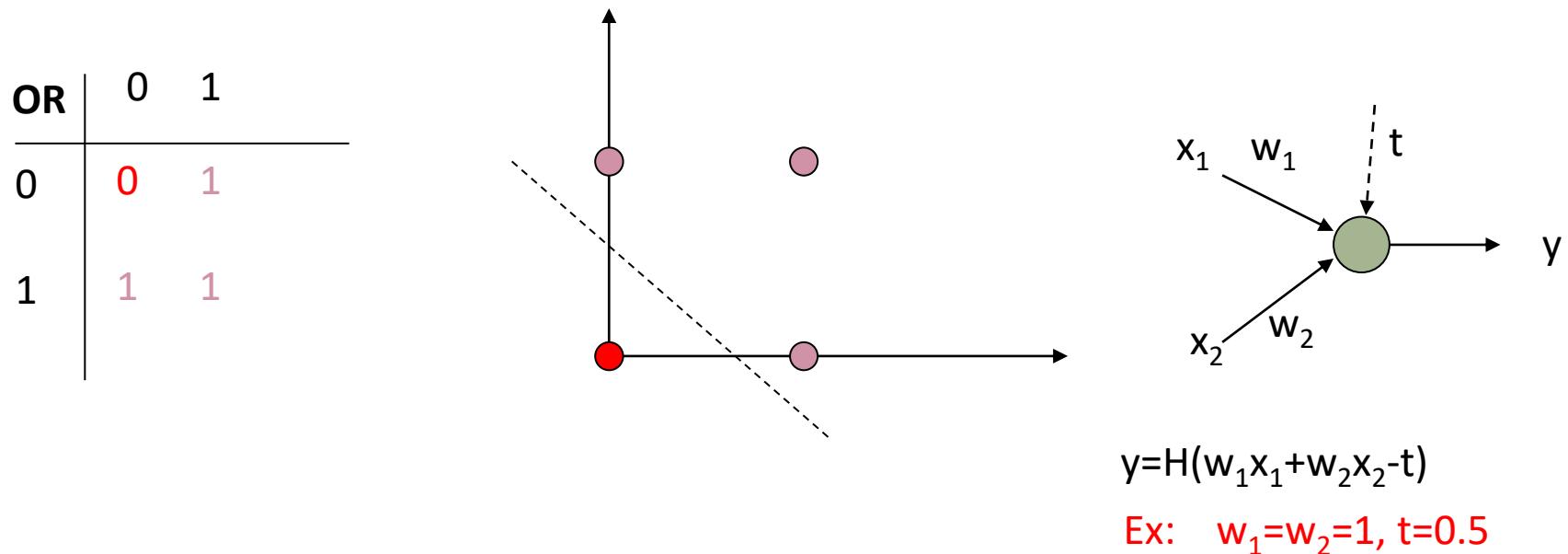
Functie sigmoidala

$$f : R \rightarrow (0,1)$$

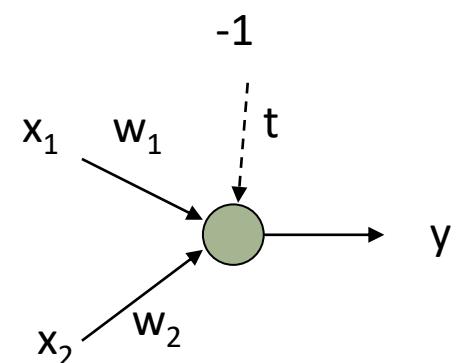
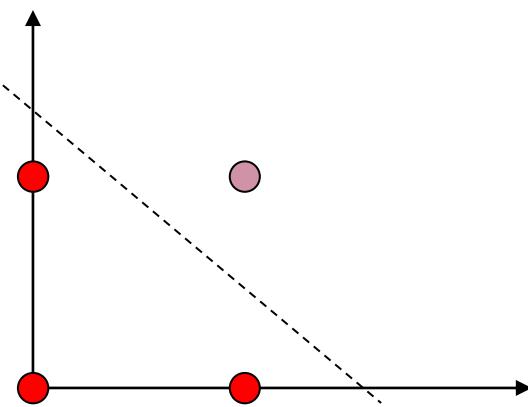
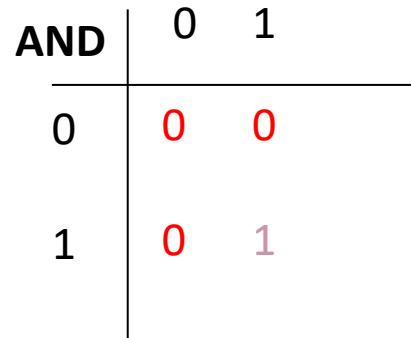
$$f(x) = \frac{1}{1 + e^{-kx}}, k > 0$$

# Ce poate face un singur neuron?

- Reprezenta (invata) functii booleene simple
- Rezolva probleme simple de clasificare binara (probleme linear separabile)



# Rezolvarea functiei AND

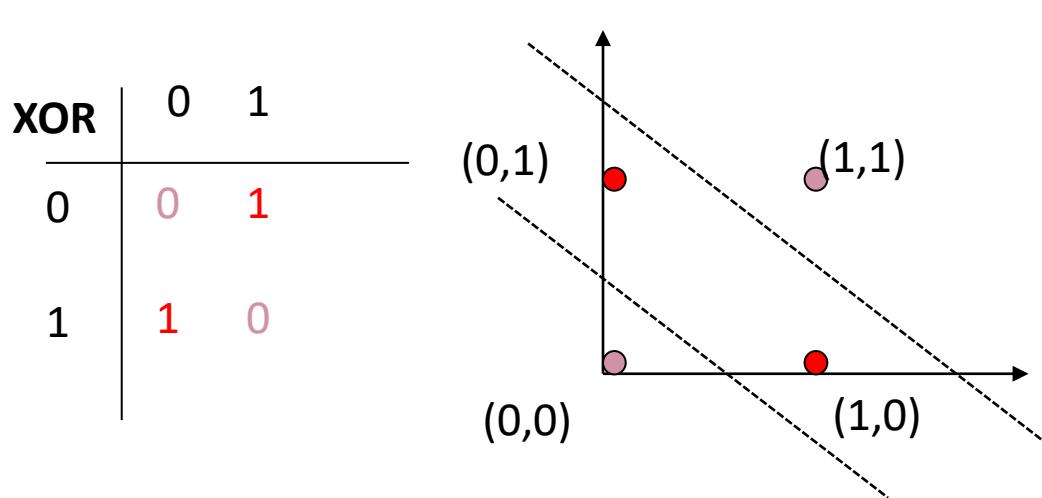


$$y = H(w_1x_1 + w_2x_2 - t)$$

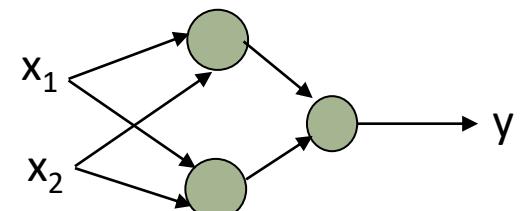
Ex:  $w_1=w_2=1, t=1.5$

# Probleme neliniar separabile

- Probleme linear separabile (ex. AND, OR): suficient o retea cu un singur nivel (cu un neuron)
- Limitele perceptronului: clase care nu sunt liniar separabile

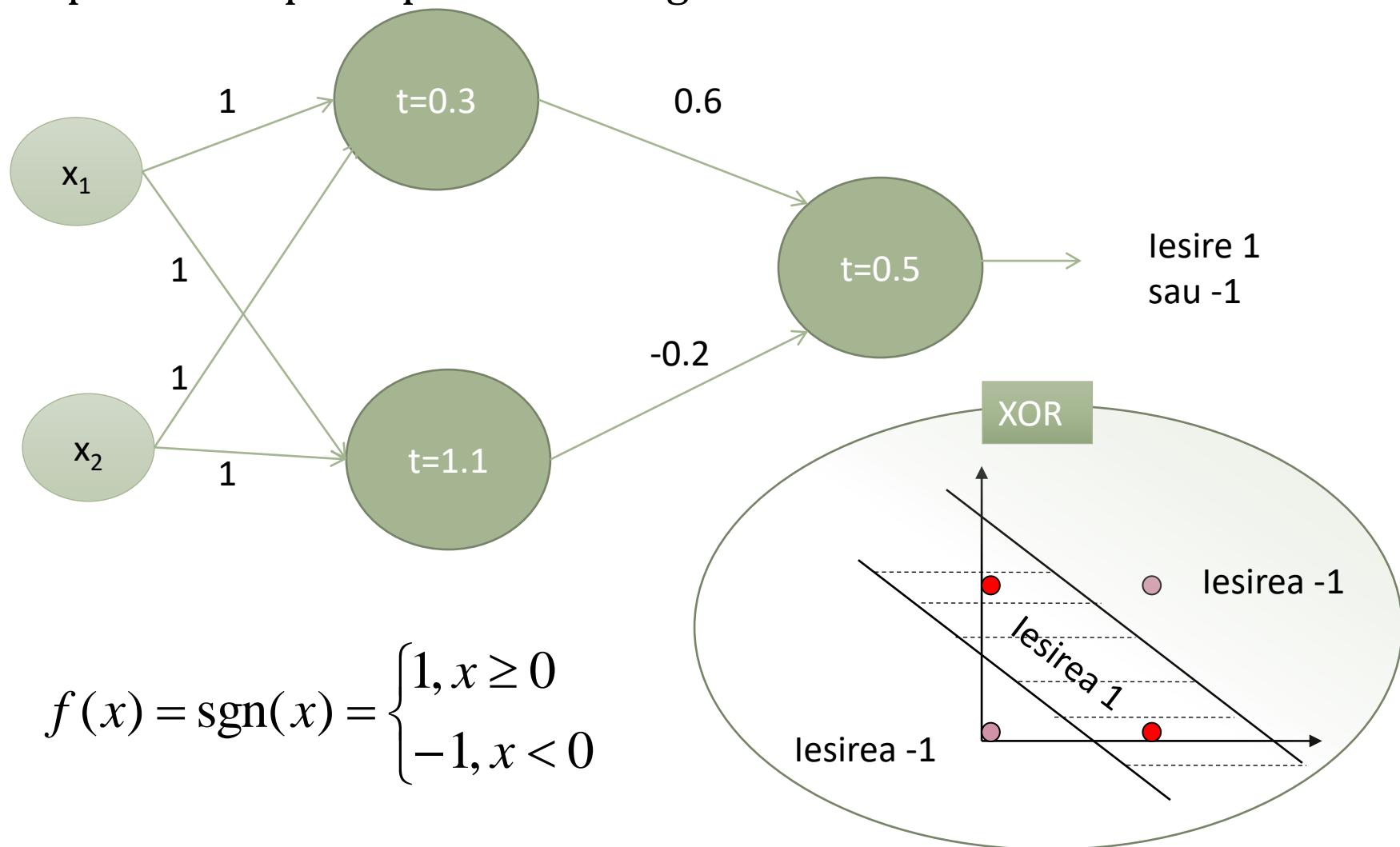


Pentru probleme neliniar separabile este necesar cel putin un nivel ascuns!



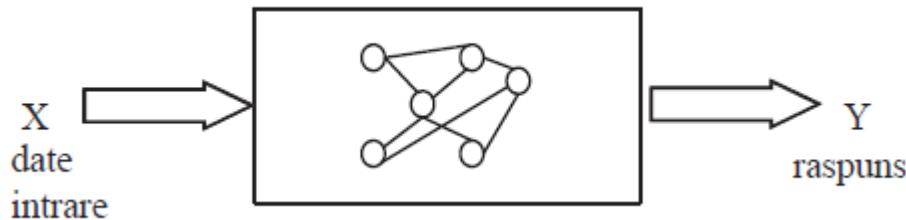
# Exemplu perceptron 2 straturi

capabil sa imparta planul in 3 regiuni de decizie



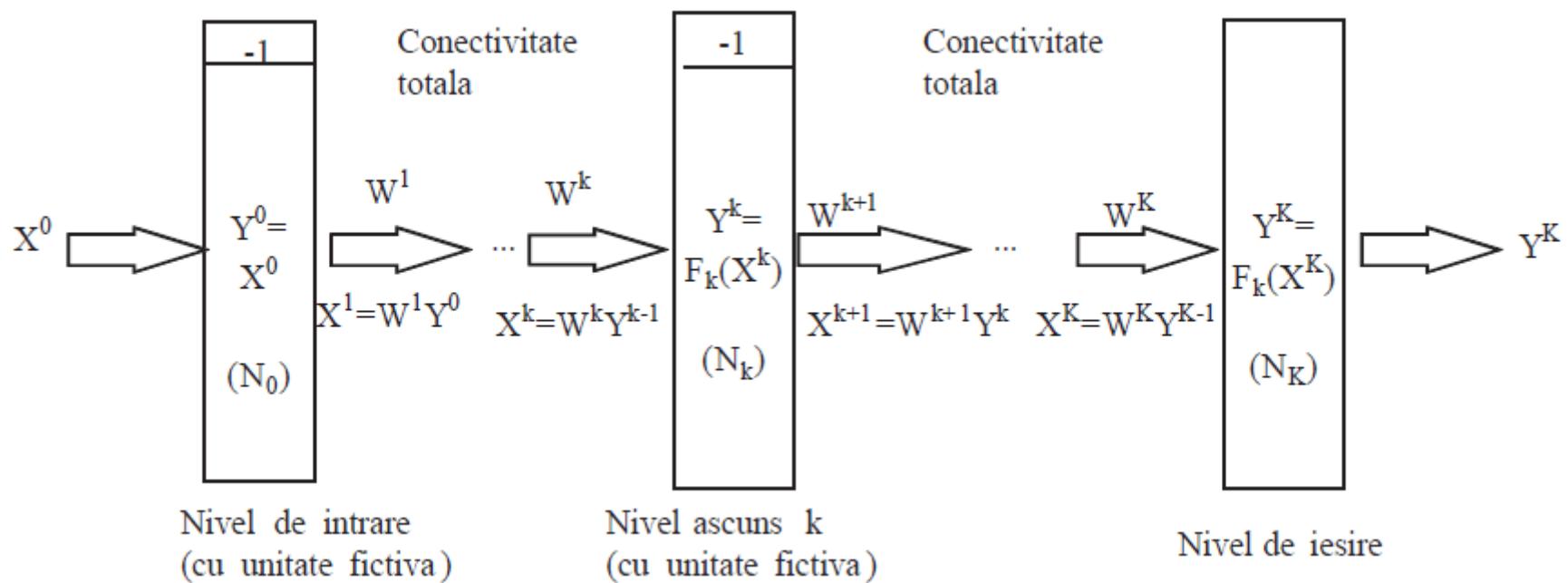
# Functionare

- Modul in care reteaua transforma un semnal de intrare X intr-un semnal de iesire Y
- Depinde de modul in care functioneaza neuronii si de cum sunt conectati
- Important: *ponderile asociate conexiunilor*

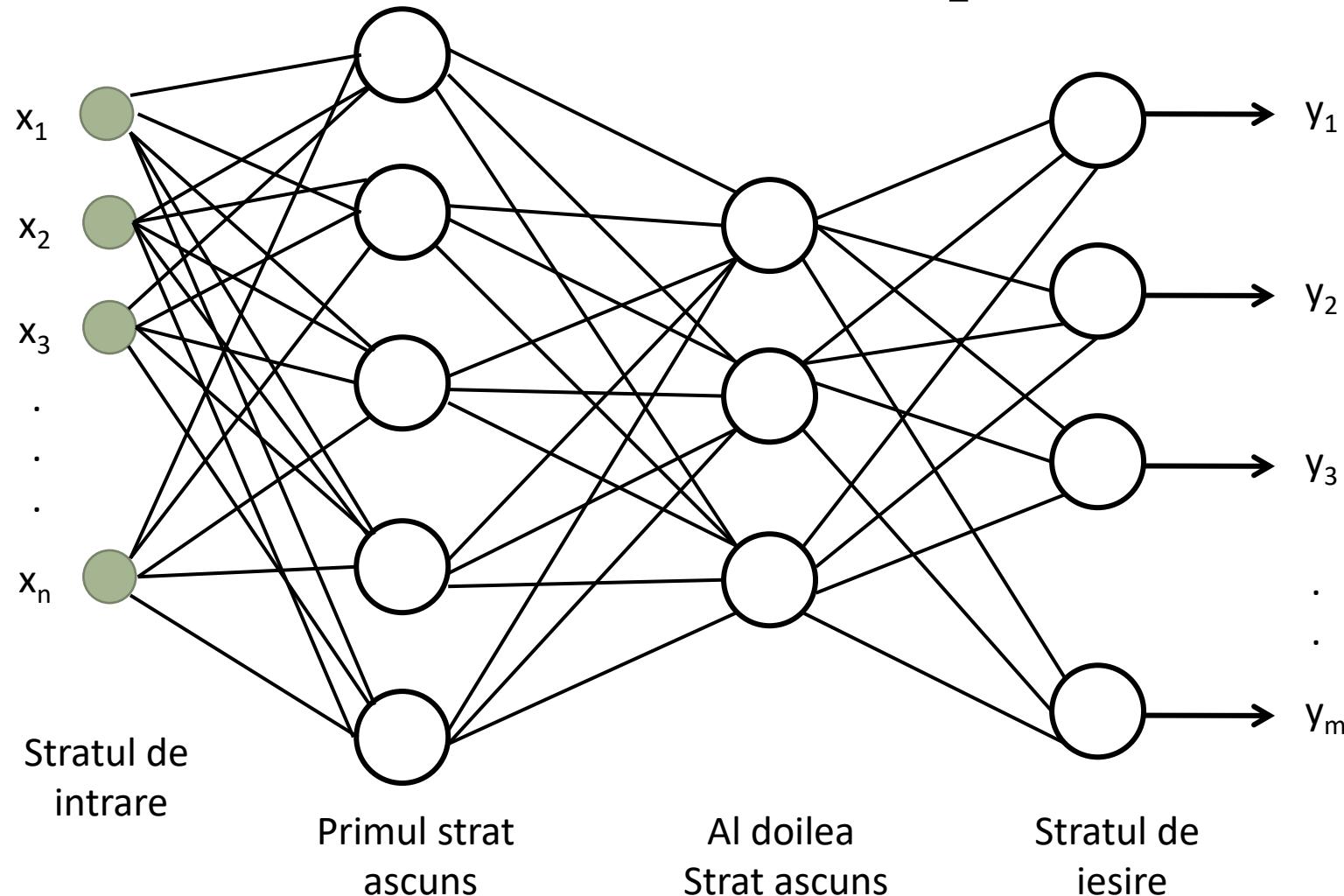


# Retele feedforward

- $X = \text{vector intrare}, Y = \text{vector ieșire}, F = \text{funcție de activare}$



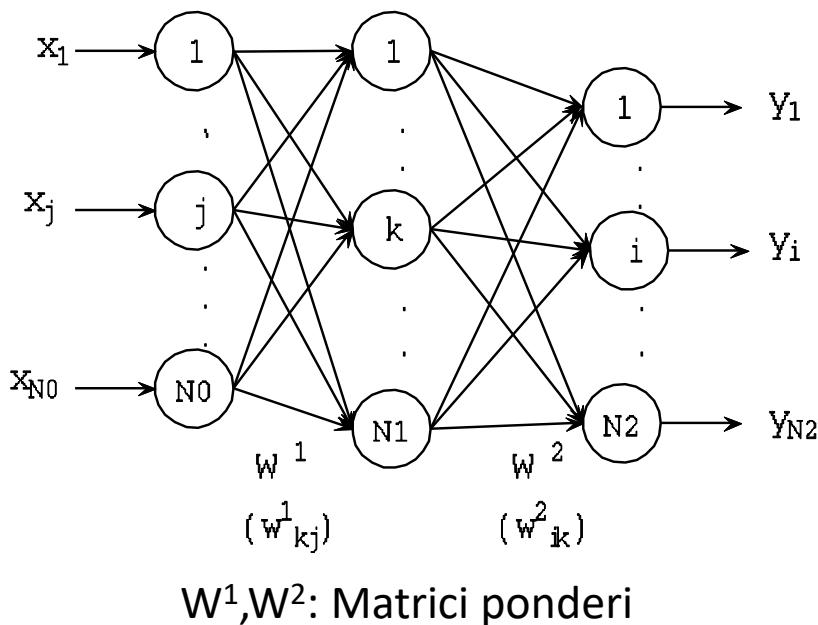
# Retele feedforward: exemplu



# Functionare Feedforward: $Y=F(X)$

- Semnalul de ieșire  $Y$  se poate obține prin aplicarea unei funcții asupra intrarilor
- Caz particular: 1 nivel ascuns
- Parametrii modelului: matricile cu ponderi  $W^1$  și  $W^2$

X: vector  
intrare



Y: vector  
ieșire

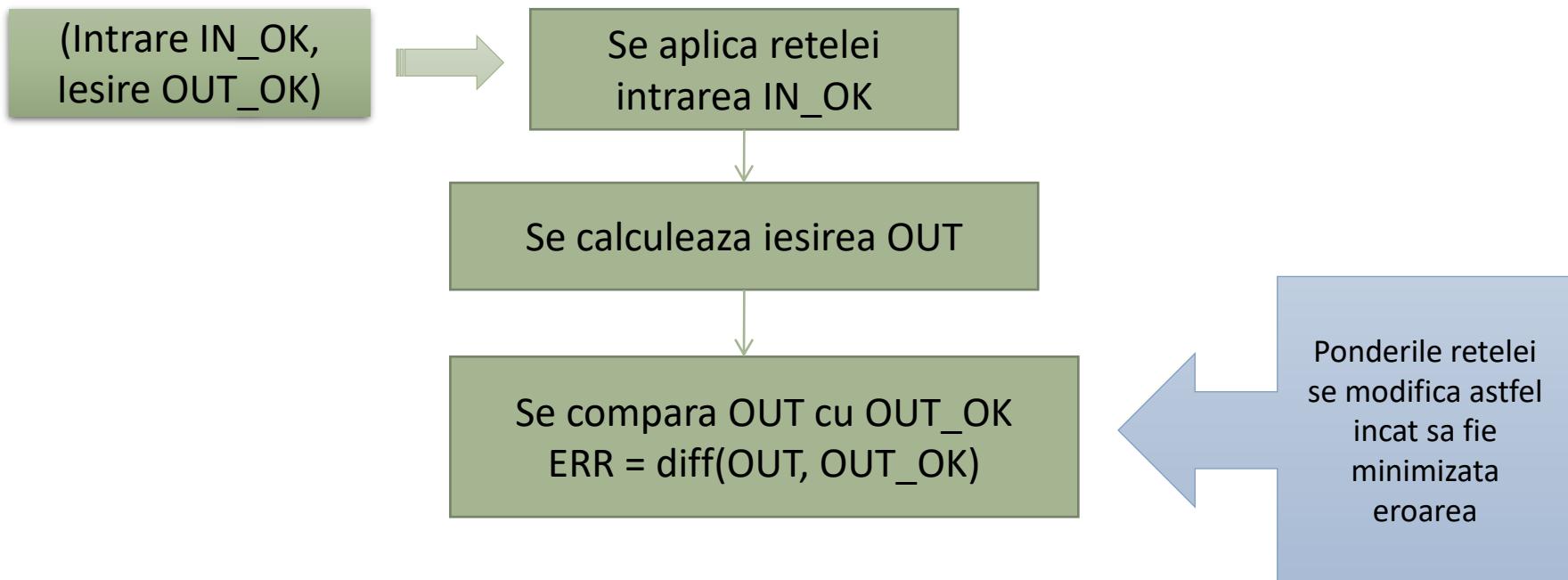
$$y_i = f_2 \left( \sum_{k=0}^{N1} w_{ik}^2 f_1 \left( \sum_{j=0}^{N0} w_{kj}^1 x_j \right) \right),$$
$$i = \overline{1, N2}$$

**Deep Learning:** rețele cu număr mare de nivele (**Deep Neural Networks**) folosite mai ales pentru recunoașterea imaginilor și a vorbirii

# Antrenarea (invatarea) RNA

- Modificarea parametrilor retelei pentru o comportare adecvata problemei
- **Scop: stabilirea valorii optime a ponderilor dintre 2 noduri**
- Capacitatea de generalizare: RNA este capabila sa produca raspunsuri la date pentru care nu a fost antrenata
- Procesul de invatare
  - Multime de informatii
  - Algoritm de adaptare la informatiile primite
- Invatare supervizata
  - Se prezinta retelei o multime de exemple de instruire (set de antrenare)
  - Se caută valorile optime ale ponderilor între oricare 2 noduri ale rețelei prin minimizarea erorii (diferența între rezultatul real  $y$  și cel calculat de către rețea)

# Invatare supervizata



- Testarea retelei antrenate: se retine din setul de antrenare un subset de testare
- Asigurarea unei bune capacitatii de generalizare: mentinerea unui nivel acceptabil de eroare pe setul de antrenare in scopul evitarii *suprainvatarii*

# Antrenarea RNA

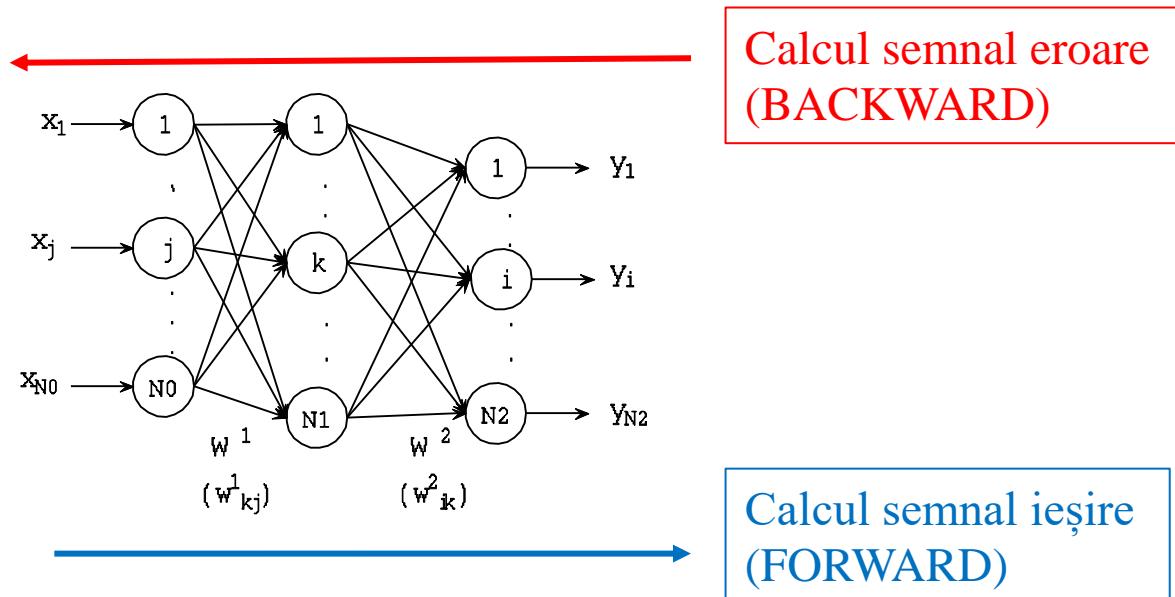
- Set de antrenare:  $\{(x^1, d^1), \dots, (x^L, d^L)\}$   
 $x$ = vector intrare,  $d$ = vector de ieșire corect
- Funcție de eroare (suma pătratelor erorilor)

$$E(W) = \frac{1}{2} \sum_{l=1}^L \sum_{i=1}^{N_l} \left( d_i^l - f_2 \left( \sum_{k=0}^{N_{l-1}} w_{ik} f_1 \left( \sum_{j=0}^{N_l} w_{kj} x_j^l \right) \right) \right)^2$$

- Scopul antrenării: minimizarea funcției de eroare
- Metode de antrenare
  - Backpropagation (propagarea inapoi a erorii)
  - Algoritmi evolutivi
  - Simulated Annealing

# Algoritmul Back Propagation

- Pentru fiecare exemplu din setul de antrenare:
  - se determină semnalul de ieșire
  - se calculează eroarea la nivelul de ieșire
  - se propagă eroarea înapoi în rețea și se reține factorul delta corespunzător fiecărei ponderi
  - se aplică ajustarea corespunzătoare fiecărei ponderi



# Algoritmul Back Propagation

Initializarea aleatoare a ponderilor: aleator din [0,1] sau [-1,1]

**REPEAT**

**FOR** i = 1, L **DO**

*Etapa FORWARD:* Se propagă informația înainte și se calculează ieșirea corespunzătoare fiecărui neuron al rețelei

$$x_k^l = \sum_{j=0}^{N_0} w_{kj}^1 x_j^l, y_k^l = f_1(x_k^l), x_i^l = \sum_{k=0}^{N_1} w_{ik}^2 y_k^l, y_i^l = f_2(x_i^l)$$

*Etapa BACKWORD:* Se stabilește și se propagă eroarea înapoi

$$\delta_i^l = f'_2(x_i^l)(d_i^l - y_i^l), \delta_k^l = f'_1(x_k^l) \sum_{i=1}^{N_2} w_{ik}^2 \delta_i^l$$

*Etapa de ajustare:* Se ajustează ponderile

$$w_{kj}^1 = w_{kj}^1 + \eta \delta_k^l x_j^l, w_{ik}^2 = w_{ik}^2 + \eta \delta_i^l y_k^l$$

# Probleme

- Viteza mica de convergenta - eroarea descreste prea incet
- Oscilatii - valoarea erorii oscileaza in loc sa descreasca in mod constant
- Minime locale - procesul de invatare se blocheaza intr-un minim local al functiei de eroare
- Supra-antrenarea si capacitatea limitata de generalizare

# Rezolvarea unei probleme -RNA

- Stabilirea **arhitecturii** initiale
- Alegerea tipului neuronilor
- Stabilirea **parametrilor ajustabili**
  - O instantiere a parametrilor  $\Leftrightarrow$  O anumita functie asociata RNA
- Algoritmul de **invatare** trebuie sa fie potrivit cu arhitectura RNA si cu cantitatea de informatie de care se dispune despre problema
- **Antrenarea** RNA pentru a rezolva o anumita problema
- **Testarea** RNA  $\Leftrightarrow$  verificarea corectitudinii raspunsurilor RNA cand primeste date de intrare care nu apartin multimii de instruire dar pentru care se cunoaste raspunsul corect
- **Utilizarea** RNA

Cursul urmator...

# Modele hibride



**BABEŞ-BOLYAI UNIVERSITY**  
Faculty of Mathematics and Computer Science



# Inteligentă Artificială

*10: Modele Hibride si Aplicatii*

**Camelia Chira**

[camelia.chira@ubbcluj.ro](mailto:camelia.chira@ubbcluj.ro)



# Am vazut deja ca...

- Algoritmi evolutivi
  - Reprezentare
  - Functia de evaluare sau de fitness
  - Operatori de variatie: incrucisare, mutatie
  - Selectia
- Algoritmi inspirati de natura (swarm intelligence)
  - Particle Swarm Optimization (PSO)
  - Ant Colony Optimization (ACO)
- Algoritmi de invatare
  - Arbori de decizie
  - Retele neuronale

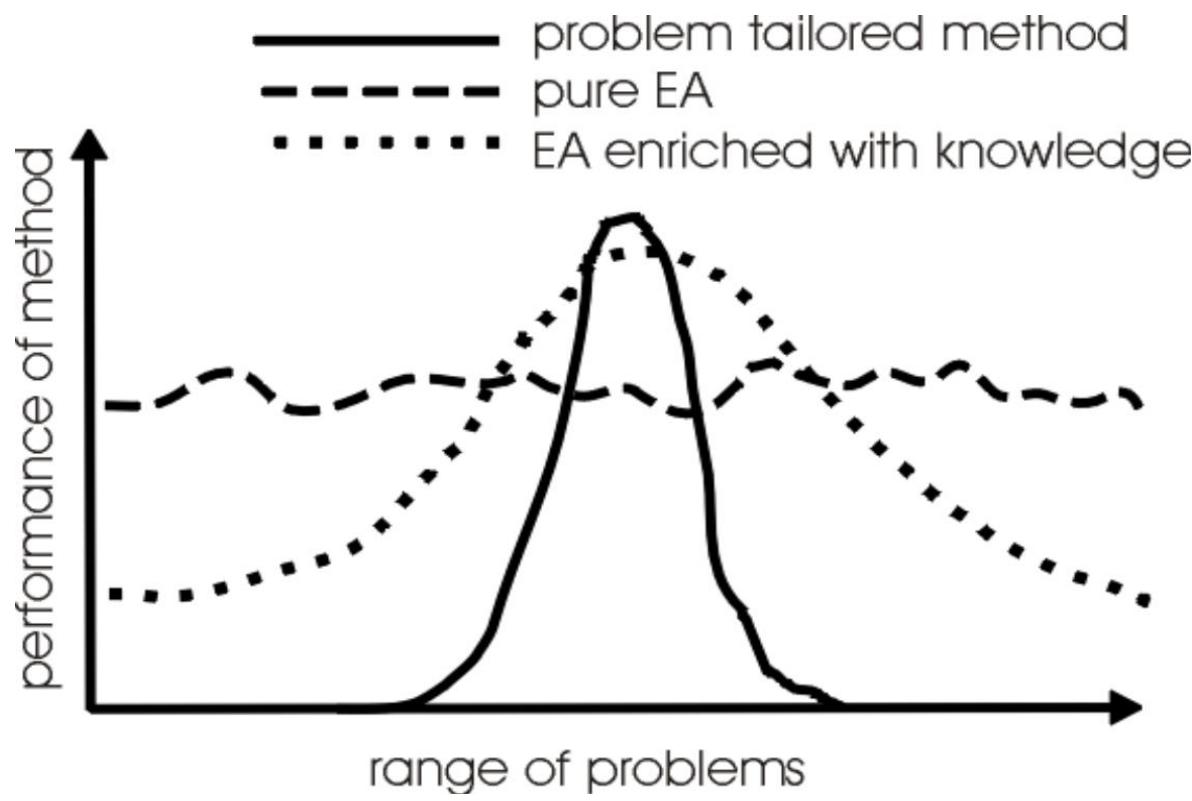
Astazi:

- Modele hibride

# Modele hibride

- **Motivatie:** Un algoritm (ex. AE) poate face parte dintr-un model / sistem mai mare
- Imbunatatirea
  - Unei tehnici existente cu elemente AE
  - Unui AE pentru o mai buna cautare a solutiilor interesante

# EA Performance



MICHALEWICZ

# Modele hibride

- Hibridizarea AE cu proceduri standard (Ex. Hill-climbing, metode de cautare greedy)
  - Populatia initiala poate fi generata folosind alte metode
  - Indivizii pot trece printr-o etapa de imbunatatire locala si apoi intra in competitie cu ceilalti indivizi din populatie
  - Etc ....**hybrid systems with fuzzy-neural-evolutionary components!**
- Exemple
  - 2-opt local search in EA pentru TSP
  - EA + Hill-climbing
  - EA + Tabu search
  - Simulated Annealing + EA
  - ES + Local search
  - Greedy crossover, local search for mutation
  - Local search + Simulated Annealing
  - etc

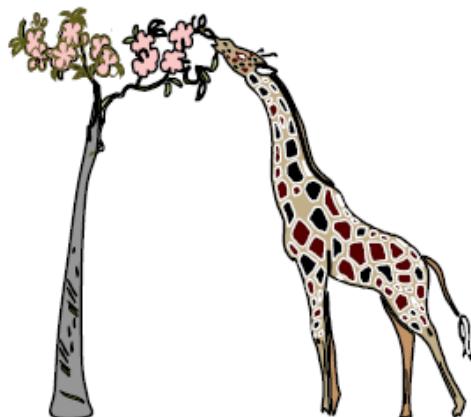
# Algoritmi memetici (Memetic Algorithms)

Algoritmi evolutivi

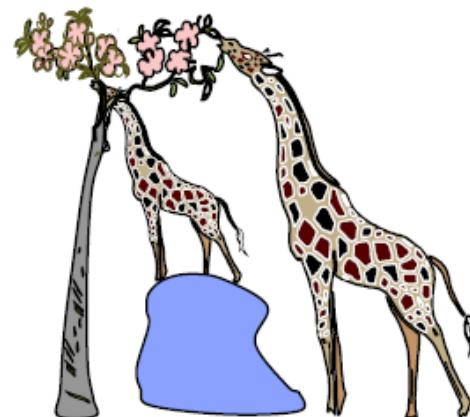


Operatori specifici de  
cautare locală

- Mai eficienti și capabili să ofere soluții de acuratețe mai bună în comparație cu EA pentru anumite probleme
- local search is applied during the evolutionary cycle

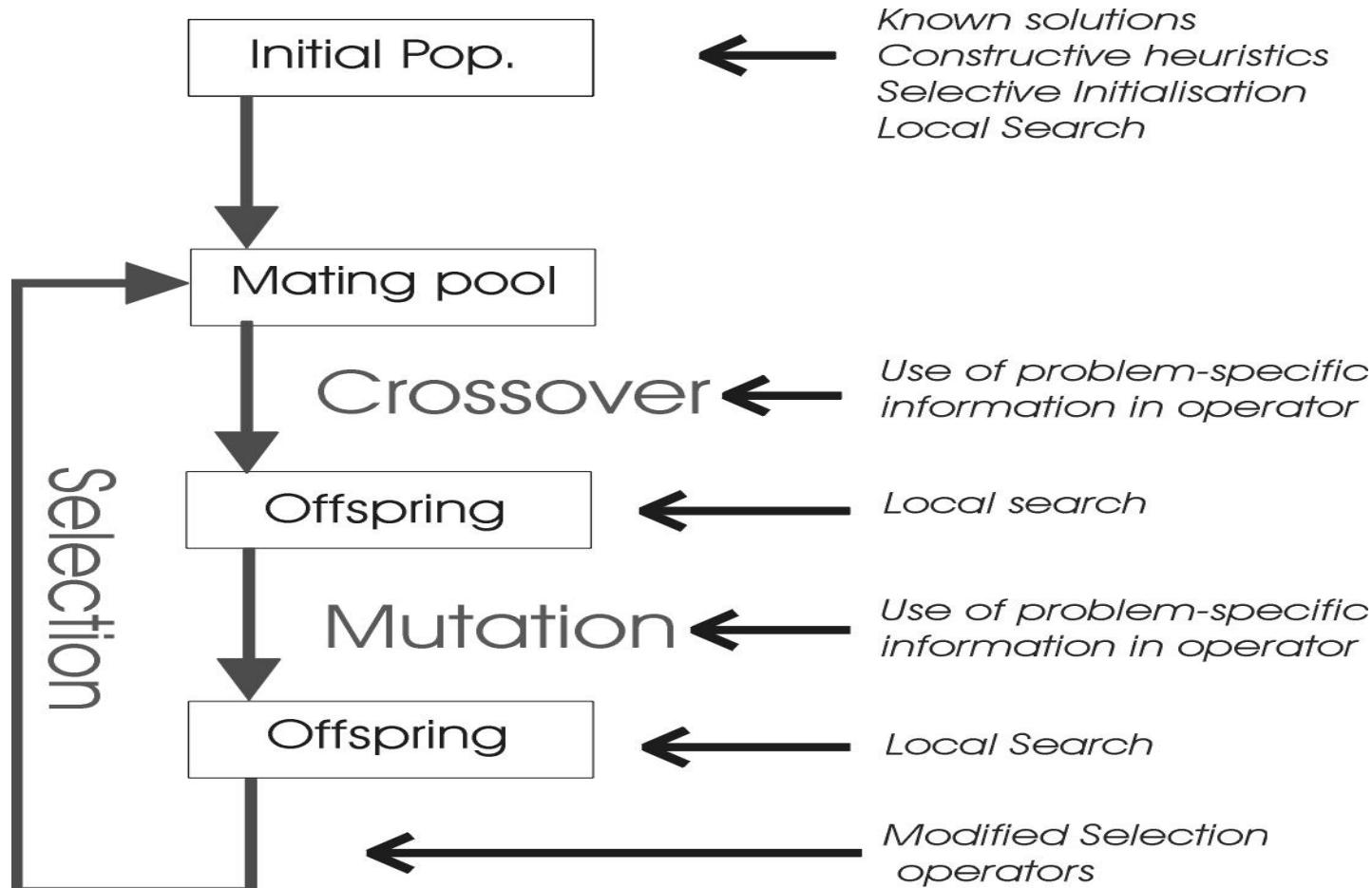


a) Genetic algorithms: Survival of the genetically fittest (i.e., tallest)



b) Memetic algorithms: Survival of the genetically fittest and most experienced

# Hibridizare – unde?



# Euristici pentru initializarea populatiei

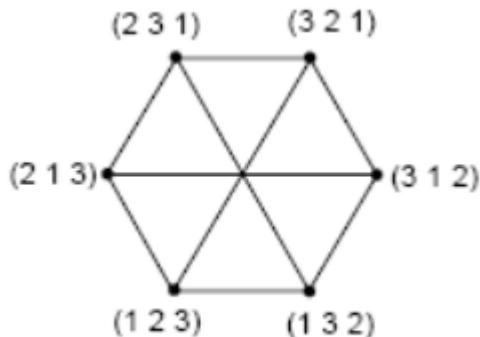
- $n$ -way tournament intre solutii generate aleator => populatia initiala
- Multi-Start Local Search
  - Alege aleator  $n$  puncte din care sa porneasca un hill-climbing
- Initializarea populatiei cu solutii deja cunoscute sau gasite folosind o alta tehnica
  - De obicei creste performanta **medie**, populatia fiind biased spre solutii cunoscute
  - **Cea mai buna** performanta vine insa tot de la solutii aleatoare (cf. unor experimente)

# Operatori specifici

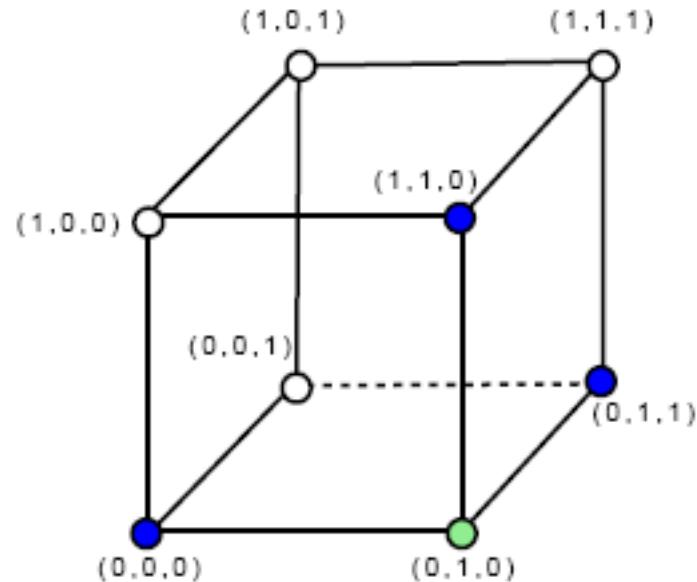
- Folosirea unor informatii specifice problemei in definirea unor operatori ‘inteligenti’
- Exemple
  - Operator specific pentru TSP: mostenirea unor subturi comune din parinti si conectarea lor folosind o euristica nearest neighbour (DPX)
  - Operatorul ‘Pull Move’ pentru modelul HP de proteine in problema de prezicere a structurii proteinelor

# Cautare locala pentru un offspring

- Poate fi privita ca si invatare de-a lungul vietii
- Stabilirea unei notiuni de vecinatate pentru fiecare solutie:  $x \leftarrow N(x)$ 
  - $N(x)$  contine toate punctele in care se poate ajunge din  $x$  printr-o **singura** aplicare a unui operator (move operator)



Vecinatate pentru permutari. Ex. Vecinii  
 $(2\ 3\ 1)$  sunt  $(2\ 1\ 3)$ ,  $(3\ 2\ 1)$ ,  $(1\ 3\ 2)$



- Nodes of the hypercube represent solutions of the problem.
- The neighbors of a solution (e.g.  $(0,1,0)$ ) are the adjacent nodes in the graph.

# Regulile cautarii locale (Pivot rules)

- Cum se efectueaza cautarea in  $N(x)$ : aleator, sistematic sau exhaustiv
- Greedy Ascent
  - Cautarea se opreste imediat ce un vecin cu un fitness mai bun a fost gasit
- Steepest Ascent
  - Toti vecinii sunt evaluati si cel mai bun este ales
- Variatii ale cautarii locale:
  - Cautarea locala se aplica intregii populatii?
    - Numai celei mai bune solutii?
    - Numai celei mai slabe solutii?
  - Cate iteratii de cautare locala pentru un individ?

# TSP: Integrarea unor metode de cautare locală

- Efort mare in a gasi reprezentari TSP + operatori de variatie
- O solutie posibila: *Hibridizari intre EAs si alte metode de cautare*

**begin**

$t \leftarrow 0$

Initializare  $P(t)$

**Aplicare cautare locală pentru  $P(t)$**

Evaluare  $P(t)$

**while** (not termination-condition) **do**

**begin**

$t \leftarrow t + 1$

Select  $P(t)$  din  $P(t-1)$

Modificare  $P(t)$

**Aplicare cautare locală pentru  $P(t)$**

Evaluare  $P(t)$

**end**

**end**

2-opt  
3-opt  
*Lin-Kernighan*  
etc  
(alți operatori single-  
sau multi-parent)

# Integrarea unor metode de cautare locala

- Putem folosi solutia imbunatatita local in recombinari

Exemplu hibridizare AE-LS:

1. Folosim 2-opt pentru a inlocui fiecare ruta din populatia curenta cu o ruta optima local
2. Indivizii care au un fitness mai bun sunt selectati mai des pentru recombinare
3. Avem incrusare si mutatie
4. Cautarea locala este aplicata fiecarui individ
5. Repeta pasii 2,3,4 pana cand o conditie de terminare e satisfacuta.

# Modele de adaptare

- **Lamarckian**

- Caracteristici dobandite de un individ in timpul vietii pot fi transmise la descendenți

Ex. Inlocuieste un individ cu vecinul mai bun

- **Baldwinian**

- Caracteristici dobandite de un individ nu pot fi transmise la descendenți

Ex. Individualul primește fitness-ul dar nu și genotipul celui mai bun vecin

**The Baldwin effect**

**MA – nu suntem contransi de realitati biologice =>**  
**Putem aplica Lamarckian learning**

# Diversitatea populatiei

- Mentinerea diversitatii poate fi o problema in EAs, mai ales cand cautarea locala tinde spre cateva solutii foarte bune
- Moduri de a induce diversitatea
  - Alege indivizii care trec prin faza de cautare locala in loc de a o aplica intregii populatii
  - Folosirea unor operatori de variatie specifici
  - Modificarea selectiei pentru a preveni duplicarile
  - Modificarea criteriului de acceptare a unui vecin in cautarea locala
  - Adaptive Boltzman Operator (Krasnogor)

# Exemplu: Boltzman MAs (Krasnogor)

- Foloseste un criteriu de acceptare inspirat Simulated Annealing: temperatura este invers proportionala cu diversitatea populatiei
- Problema de maximizare
- $\Delta f = \text{fitness vecin} - \text{fitness curent}$

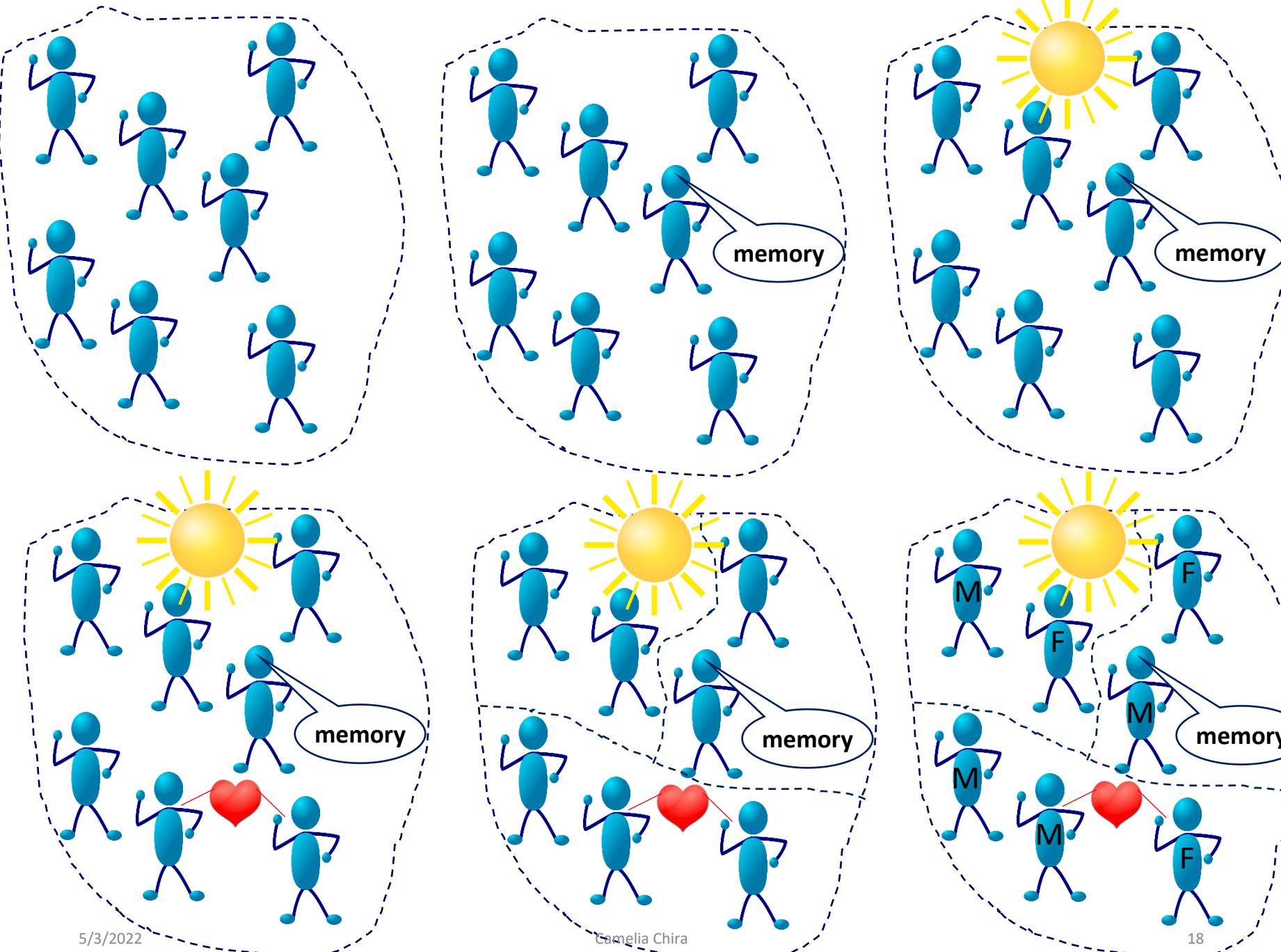
$$P(\text{accepting neighbour}) = \begin{cases} 1 & \Delta f > 0 \\ e^{\frac{k\Delta f}{f_{\max} - f_{\text{avg}}}} & \Delta f < 0 \end{cases}$$

Populatie diversa => fitness divers, temperatura mica, accepta numai vecini care imbunatesc fitness-ul => *Exploitation*

Populatia converge => temperatura mare, probabilitate mai mare de a accepta si vecini mai slabii => *Exploration*

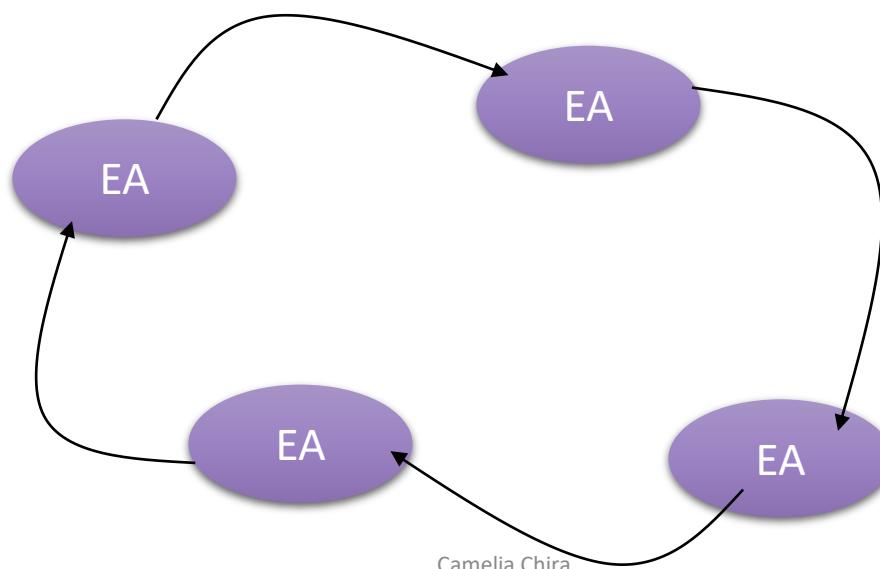
# Alagerea operatorilor in MA

- Folosirea unui operator in cautare locala **diferit** de cel folosit pentru incrucisare si mutatie prezinta avantaje teoretice (Krasnogor, 2002)
- Folosirea mai multor operatori de cautare locala si stabilirea unui mecanism prin care sa se aleaga unul din ei
- Integrarea unor mecanisme de invatare si adaptare pentru alegerea unui operator
- Folosirea unor operatori din alti algoritmi folositi deja pentru problema abordata (ex. 2-opt pentru TSP)
- Integrarea unor cunostinte specifice problemei
- *Idei*
  - Folositi mai multi operatori de cautare locala in tandem
  - Adaugati o pozitie in individ care indica ce operator de cautare locala se foloseste (aceasta gena este mostenita de la parinti, si poate fi supusa mutatiei)



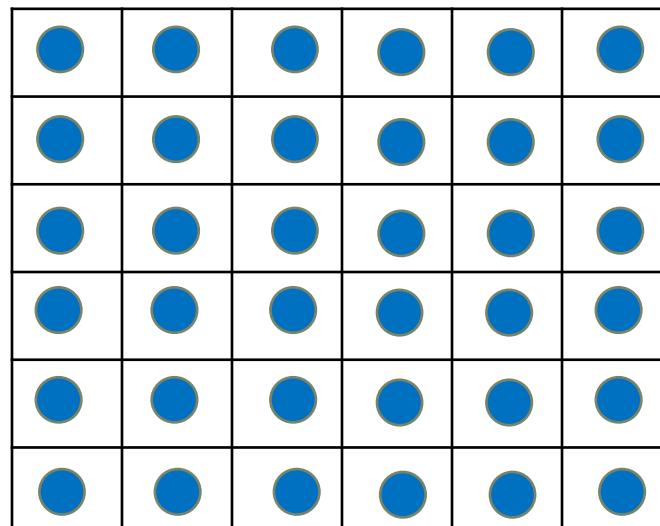
# Posibilitati...ISLAND MODEL

- Putem impari populatia in mai multe subpopulatii => **island, migration or coarse-grain models**
- Putem rula EAs diferiti pe fiecare subpopulatie (variatia si selectia la nivel de subpopulatie)
- Unul sau multi indivizi migreaza din cand in cand intre subpopulatii



# Posibilitati...DIFFUSION MODEL

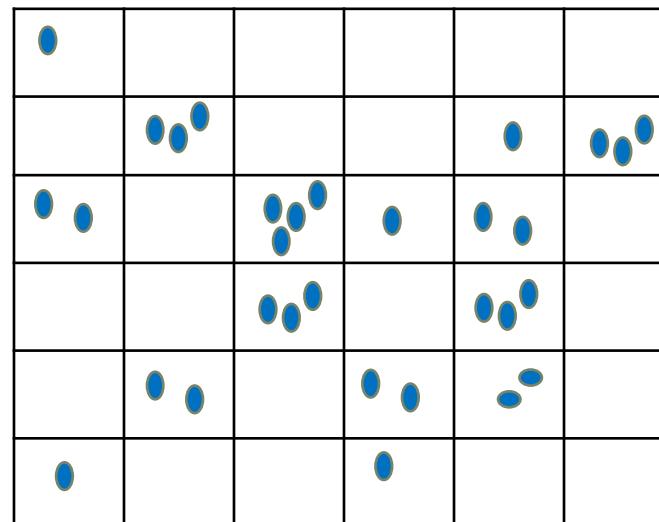
- Putem organiza populatia prin repartizarea fiecarui individ pe o zona geografica=> **diffusion, neighborhood or fine-grain models**
- Fiecare inivid are o vecinatate data de topologie
- Selectia, variatia si inlocuirea parintilor sunt restrictionate de topologie



Topologie grid intr-un diffusion model clasic

# Posibilitati...PATCHWORK MODEL

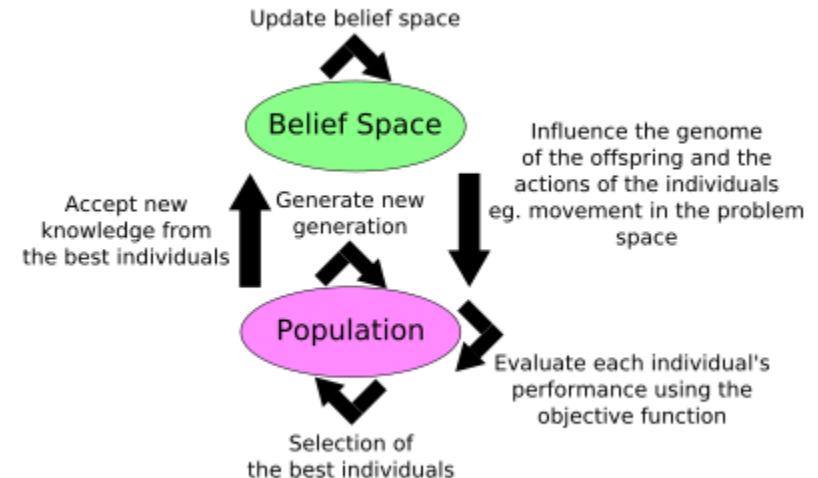
- Combina **island** and **diffusion models**
- **Patch** = fiecare celula din grid
- Fiecare inivid este modelat ca un agent intr-o nisa care interactioneaza cu mediul si deciziile se bazeaza pe informatii locale
- Proprietati ale unui individ: maximum life span, ability to breed, mortality, preferences in decisions



Artificial  
Life

# Cultural Algorithms

- Doua moduri de mostenire:
  - Microevolutionary level: trasaturi genetice si de comportament
  - Macroevolutionary level: credintele (beliefs) populatiei
  - Cele doua moduri interactioneaza printr-un canal de comunicatie care le permite indivizilor sa schimbe credintele si sa influenteze comportamentul individual pe baza credintelor populatiei
- *“evolution of evolution”*  
(indivizii din populatie invata in timpul vietii)
  - Evolutie: invatare la nivel de specie, invatare sociala in societati umane



# Sumar hibridizari AE

- AEs ofera o abordare extrem de flexibila in rezolvarea problemelor
- Pot fi usor hibridizati cu algoritmi traditionali (ex. Metoda greedy in initializarea populatiei) sau pot fi extinsi pentru a include diferite aspecte observate in natura
- Idei: *populatii multiple, memorie, invatare individuala, invatare sociala, etc*
- **No Free Lunch Theorem**
  - Nici un algoritm nu este cel mai bun pentru orice problema.
  - *Indiferente de ce extensii si modificari i se aduc unui algoritm, vor fi intotdeauna probleme pentru care este foarte bun si altele pentru care merge extrem de rau*

# Hibridizare SI models

- **Metaeuristica ACO**

**procedure ACO**

**while (not-termination-criterion)**

**schedule sub-procedures**

*generate-&-manage-ants()*

*update-pheromones()*

*execute-daemon-actions()*

**end schedule sub-procedures**

**end while**

**end procedure**

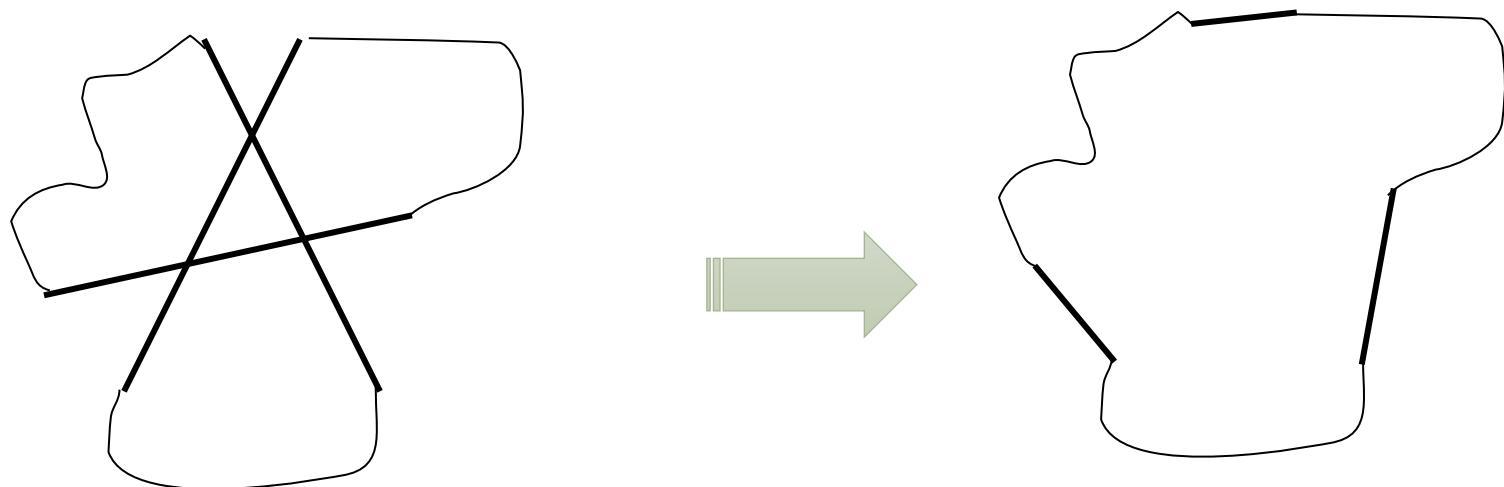
**... + Local Search**

# ACS + Local Search

- **REPEAT**
  - Fiecare furnica este pozitionata intr-un nod de start
  - **REPEAT**
    - **State transition rule:** Fiecare furnica construieste incremental o solutie
    - **Local pheromone updating rule**
  - **UNTIL** Toate furnicile - un drum complet
  - Fiecare furnica - **Local Search (cautare locală)**
  - **Global pheromone updating rule:** Urmеле de feromon sunt modificate la nivel global
- **UNTIL** STOP condition

# ACS-3-opt = ACS + Cautare locala cu 3-opt

- La fiecare iteratie ACS, pentru fiecare furnica
  - 3 muchii odata sunt schimbate iterativ pana cand un optim local este atins (toate celelalte muchii sunt pastrate)
- Restricted 3-opt (pentru ATSP)
  - Ordinea in care orasele sunt vizitate nu se schimba
  - $(k,l), (p,q), (r,s) \rightarrow (k,q), (p,s), (r,l)$



# ACS + cautare locală

- Cautarea locală este complementara mecanismului stigmergic
- **Calitatea** soluțiilor este **imbunatatita**
- Cautarea locală porneste de la solutii semnificative generate de furnici
- Paralela cu algoritmi evolutivi:
  - Incrucisare – feromon
  - Mutatie – cautare locală
- **Costul computational creste**

# Aplicatii

- **Optimizare numerica**
- **Probleme de optimizare combinatoriala**
  - Problema rucsacului
  - Traveling Salesman Problem
  - Vehicle Routing Problem
  - ...
- **Alte probleme de optimizare**
  - Detectarea comunitatilor in retele complexe
  - Detectarea de reguli pentru automate celulare
  - Detectarea structurii proteinelor
  - Determinarea rutelor in trafic
  - ...

# Probleme de optimizare combinatorială

- **Problema rucsacului (0/1 Knapsack Problem)**

- Instante:  
[https://people.sc.fsu.edu/~jburkardt/datasets/knapsack\\_01/knapsack\\_01.html](https://people.sc.fsu.edu/~jburkardt/datasets/knapsack_01/knapsack_01.html)
- Reprezentare binara
- Operatori de incruisare si mutatie specifici codificarii binare
- Functia de fitness: cu penalizare

- **Problema comis-voiajorului (Travelling Salesman Problem)**

- Instante: <http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/>
- Codificare prin permutari
- Operatori de incruisare si mutatie specifici permutarilor

# Probleme de optimizare combinatorială

- **Vehicle Routing Problem**

- Instante: <https://sites.google.com/site/vrphlibrary/benchmark-problems>

- **Def:** given a fleet of vehicles with uniform capacity, a common depot, and several customer demands, finds the set of routes with overall minimum route cost which service all the demands

- *Avem un depozit și mai multe orașe, fiecare cu o anumita cerere*
    - *Avem un camion care are o anumita capacitate*
    - *Trebuie să gasim rutile de cost minim de la depozit care să viziteze toate orașele (tinând cont de restricții)*

- Reprezentare prin permutări:

**2 4 9 6 7 8 3 1 5**

- route1: [0 2 4 9 0]
    - route2: [0 6 7 8 3 0]
    - route3: [0 1 5 0]

- Operatori de incruisare și mutație specifici

# Alte probleme de optimizare

- Detectarea comunitatilor in retele complexe  
*(Community detection in complex networks)*
- Detectarea de reguli pentru automate celulare  
*(Density classification in Cellular Automata)*
- Detectarea structurii proteinelor  
*(Protein structure prediction)*
- Probleme de optimizare a rutelor  
*(Route Optimization)*
- Determinarea rutelor in trafic  
*(Traffic Assignment Problem)*

# Complex Networks

- ***Social networks***

- acquaintance networks, collaboration networks

- ***Technological networks***

- the Internet, the Worldwide Web, power grids

- ***Biological networks***

- Neural networks, food webs, metabolic networks

- **Degree distribution**

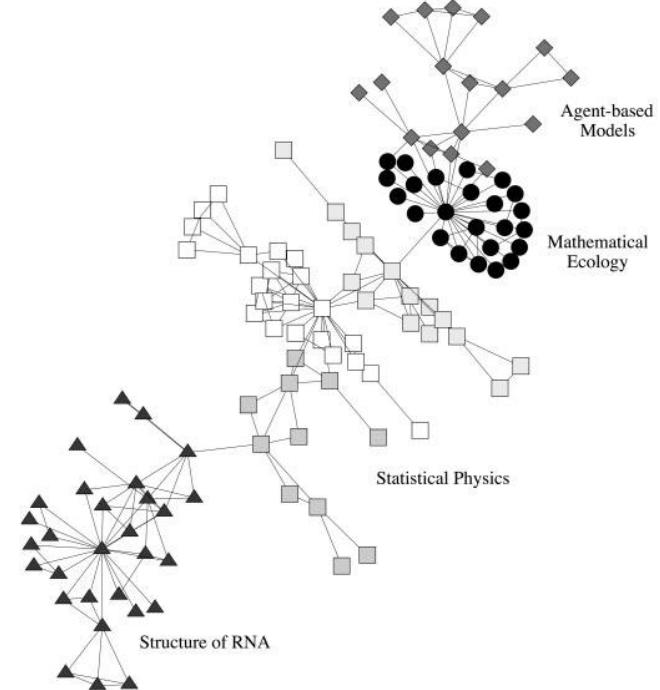
- typically many vertices in a network with low degree and a small number with high degree

- **Small-world effect**

- the average distance between vertices in a network is short (13, 14)

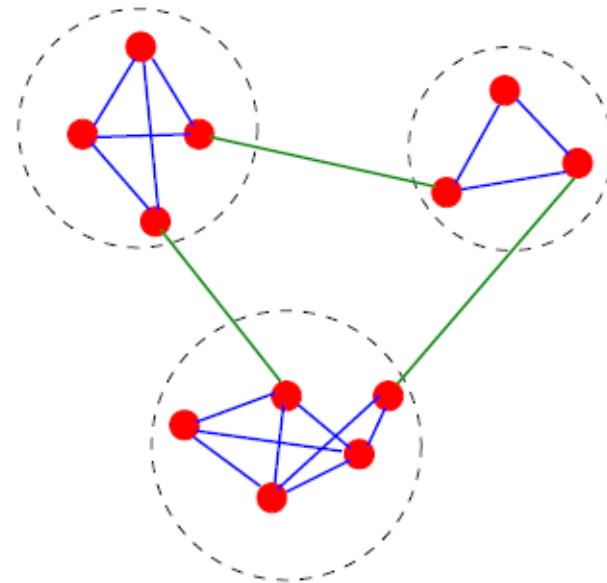
- **Network transitivity**

- two vertices both neighbors of the same third vertex have a high probability of being neighbors of one another



# Community Detection Problem

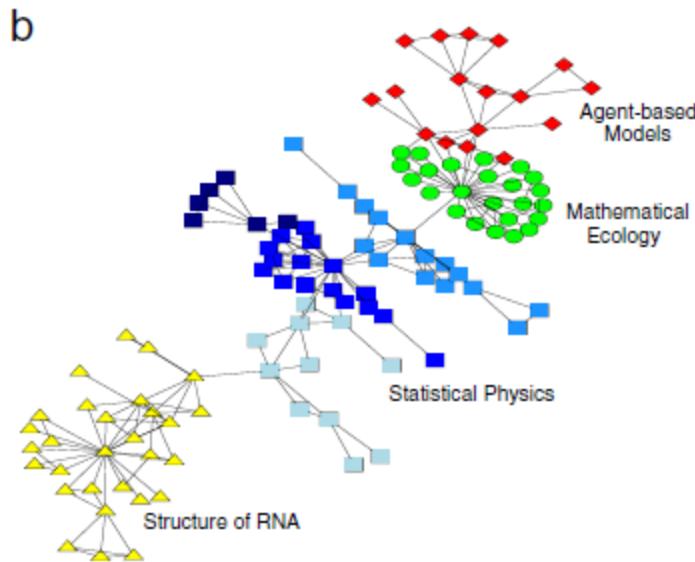
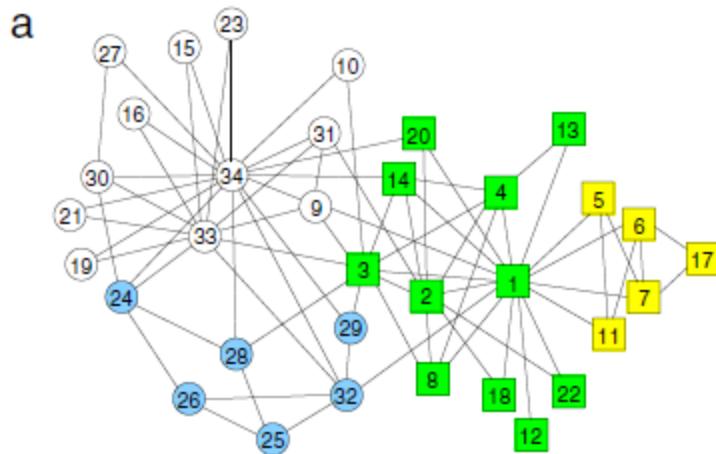
- A **community** in a network is a group of nodes densely connected but sparsely connected with the nodes belonging to other communities



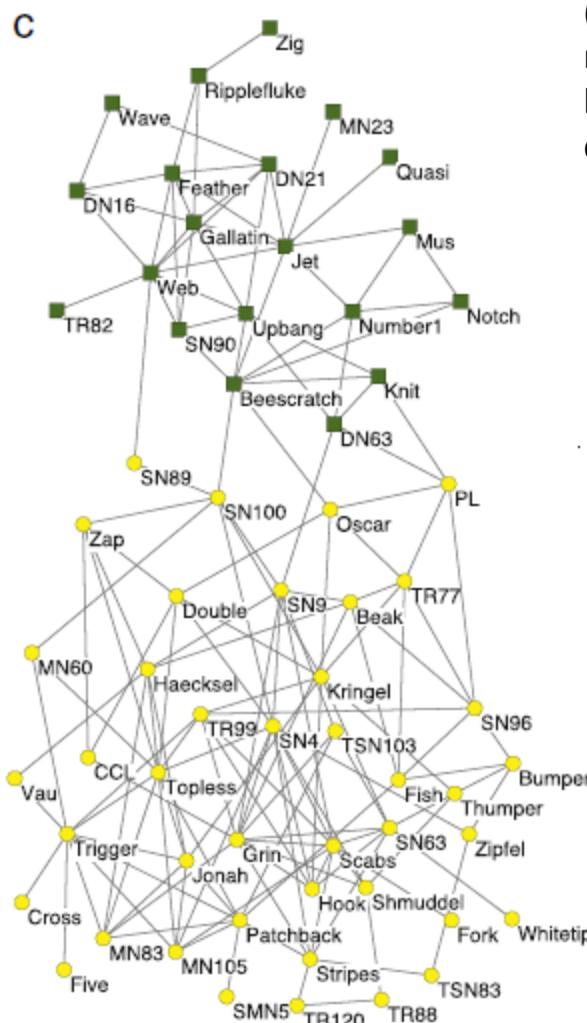
## Def. **The *community structure***

- *the division of networks into groups* (also called clusters) having dense intra-connections, and sparse inter-connections

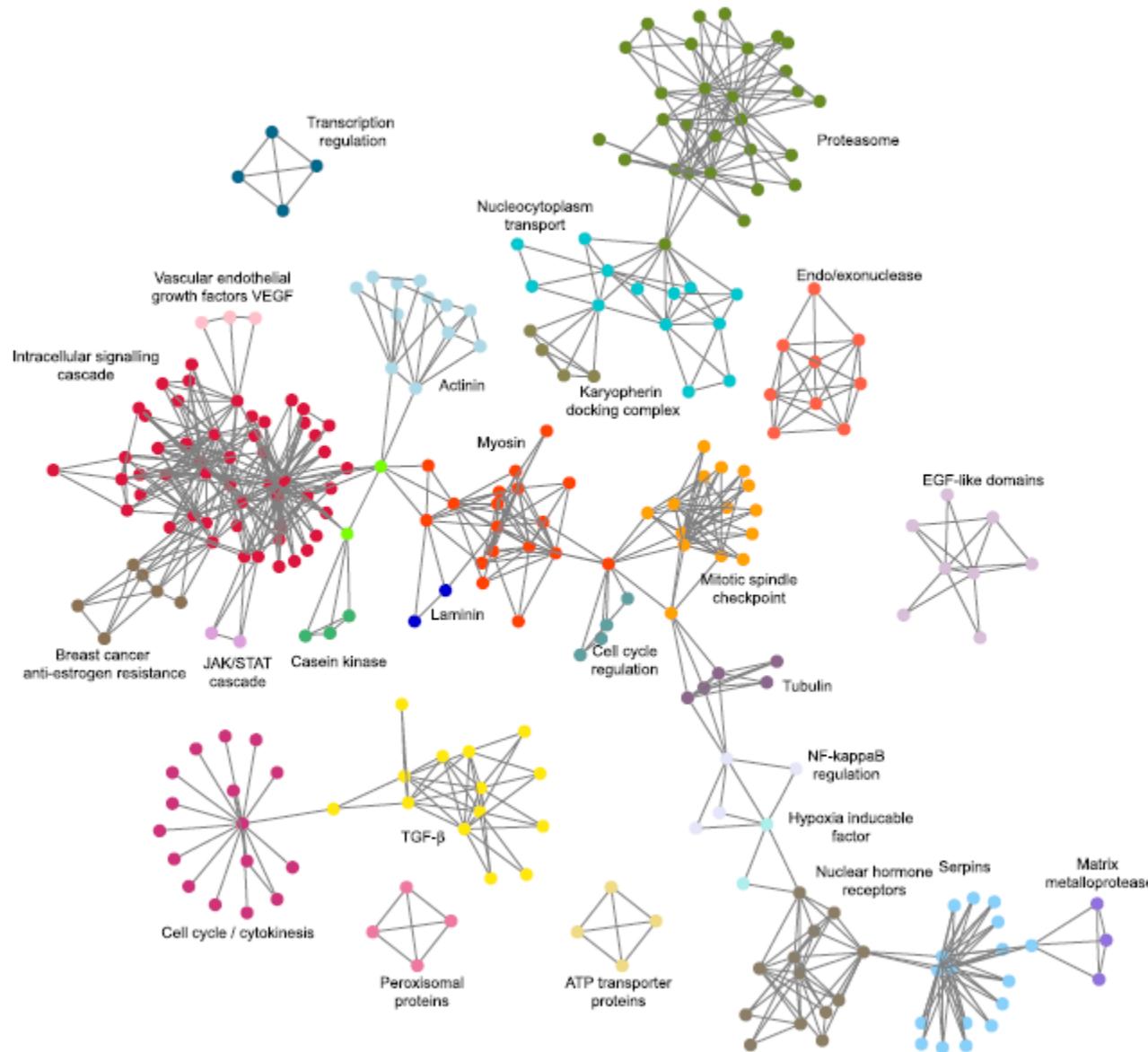
(a) Zachary's karate club, a standard benchmark in community detection



(b) Collaboration network between scientists working at the Santa Fe Institute.



(c) Lusseau's network of bottlenose dolphins.



S. Fortunato / Physics Reports 486 (2010) 75–174

Community structure in protein-protein interaction networks - the interactions between proteins in cancerous cells of a rat.

# Definitions

- A social network: modeled as a graph  $G = (V, E)$ 
  - where  $V$  is a set of objects, called nodes or vertices,
  - and  $E$  is a set of links, called edges, that connect two elements of  $V$ .
- **The adjacency matrix** (for  $N$  nodes):
  - the  $N \times N$  matrix  $A$
  - *the entry at position  $(i, j)$  is 1 if there is an edge from node  $i$  to node  $j$ , 0 otherwise*

**Community detection:** Problem of detecting  $k$  communities in a network :

- Find a partitioning of the nodes in  $k$  subsets that are highly intra-connected and sparsely inter-connected.

*OR*

- Find a partitioning of  $A$  in  $k$  sub-matrices that maximize the sum of densities of the sub-matrices.

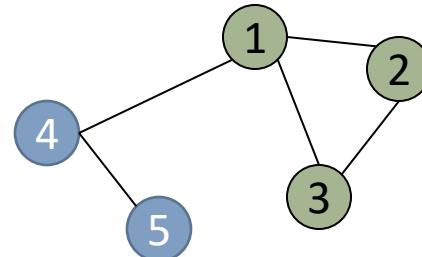
# Representation 1

1 <sup>st</sup> node's community ID →	12
2 <sup>nd</sup> node's community ID →	29
3 <sup>rd</sup> node's community ID →	90
	12
.	.
.	.
.	.
.	.
.	.
.	.
.	.
.	.
.	.
(n-1) <sup>th</sup> node's community ID →	1
n <sup>th</sup> node's community ID →	3
	12
	4

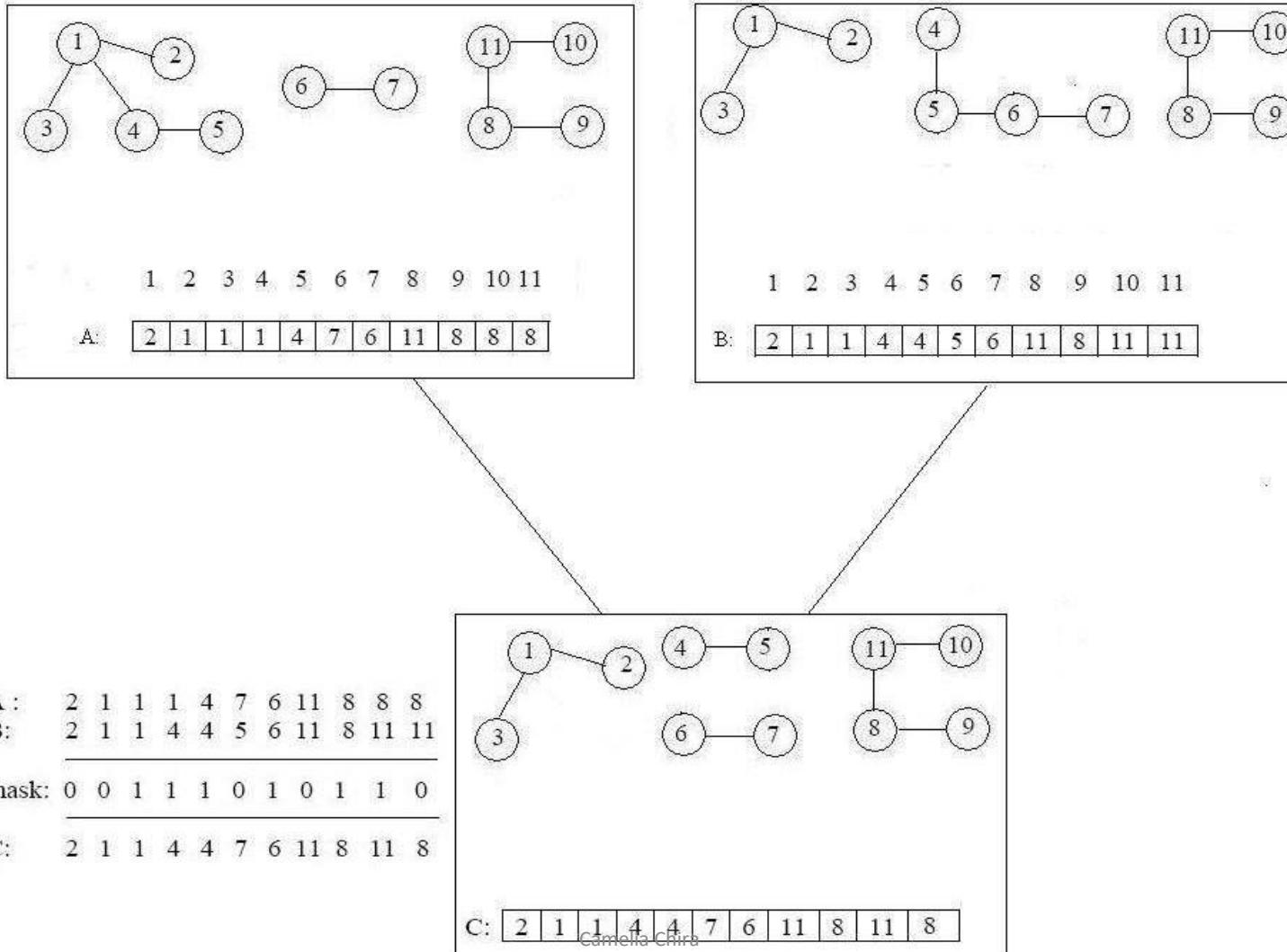
## Representation 2

- Every individual is represented as an integer array of size  $N$ , where  $N$  is the number of nodes in the network
- Each position  $i$  in the array assumes a value  $j$  (where  $j$  can be an integer number from 1 to  $N$ ) which is translated to a partitioning in which nodes  $i$  and  $j$  belong to the same cluster

1	2	3	4	5
3	1	2	5	4



# Crossover



# Fitness Function

- Newman Modularity measure

$$Q = \sum_i (e_{ii} - a_i^2)$$

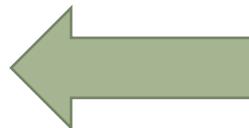
- C. Pizzuti, PPSN 2008

- $S = (I, J)$  sub-matrix of  $A$

$$a_{iJ} = \frac{1}{|J|} \sum_{j \in J} a_{ij}, \text{ and } a_{IJ} = \frac{1}{|I|} \sum_{i \in I} a_{ij}$$

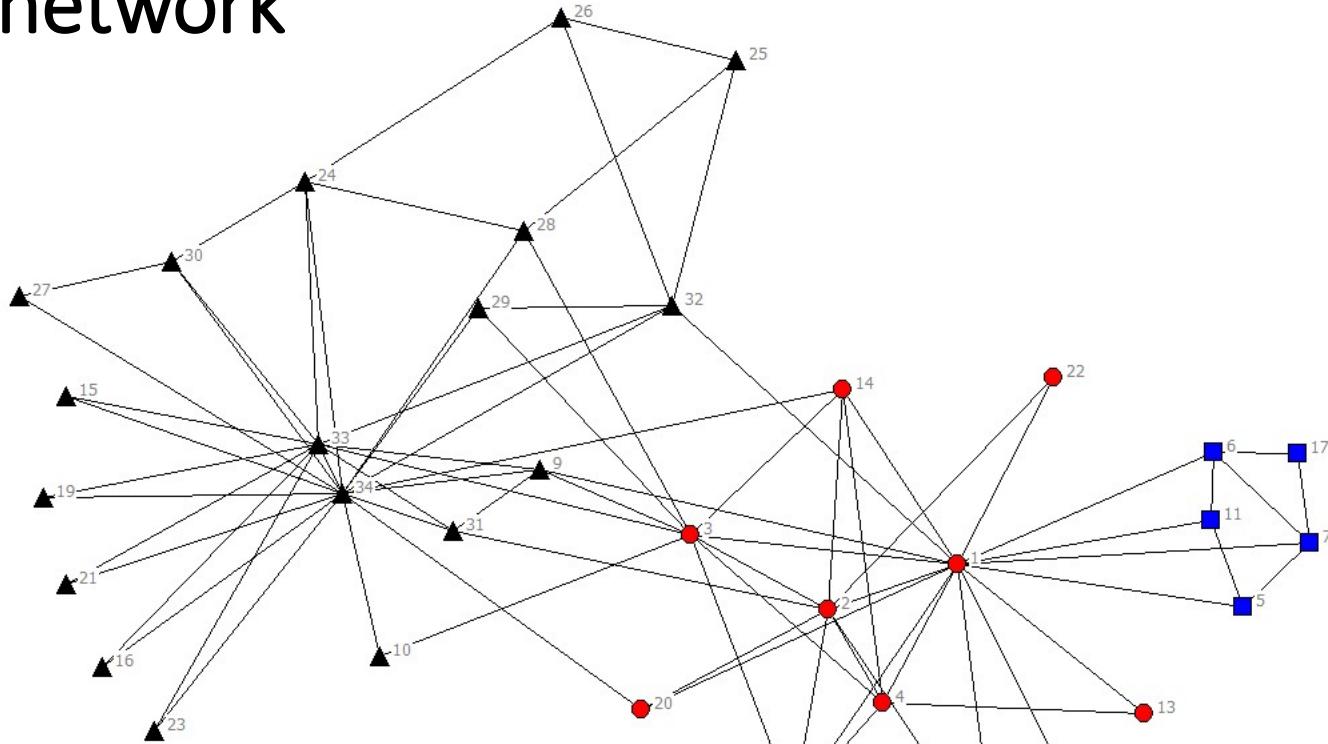
$$\mathbf{M}(S) = \frac{\sum_{i \in I} (a_{iJ})^r}{|I|}$$

$$\mathcal{CS} = \sum_i^k Q(S_i)$$



Community Score

# Karate network



Chromosome value = 10 17 0 2 10 10 16 2 30 33 5 0 0 1 33 33 33 5 0 32 0 32 0 32 32 32 25 31 33 23 33 26 32 33 31 31

Cluster 0 : 1 3 4 8 12 13 20 22 2 18 14

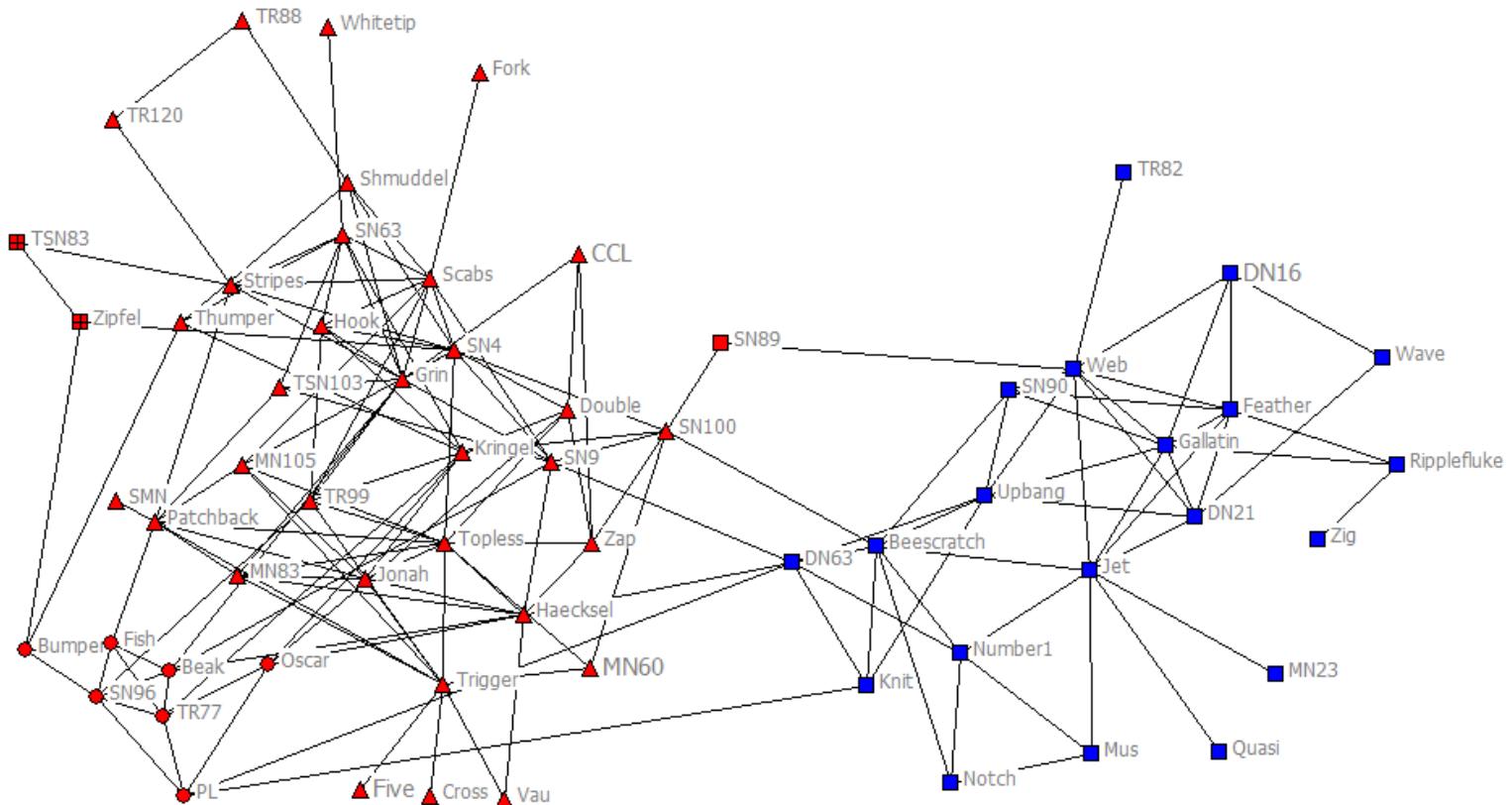
Cluster 1 : 5 11 6 7 17

Cluster 2 : 9 31 33 32 19 21 23 24 28 10 34 15 16 27 29 30 25 26

Newman modularity for global best: 0.39907955292570685

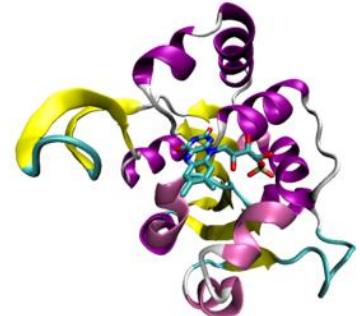
Normalized Mutual Information for global best: 0.8255181611085524

# Community structure for dolphins network with NMI=1

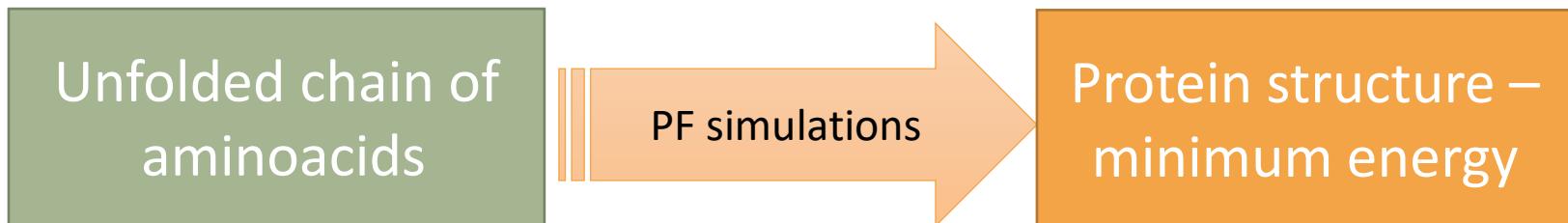


The behavior of 62 dolphins over a period of 7 years. The number of edges is 159, each emphasizing a statistically significant frequent association.

# Protein Folding



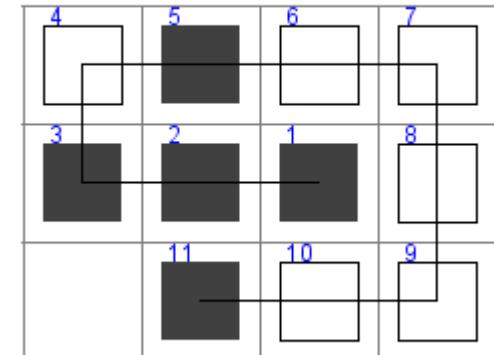
- The structure of a protein determines its function
  - The knowledge generated by a correct prediction of protein tertiary structures is of huge importance for a better treatment of certain diseases and can provide significant insights in the structure-based drug design field
- *“If a protein is to find its functional conformation space by wandering randomly through conformation space, an excess of  $10^{50}$  years would be required for folding”  
**(Levinthal paradox)***



# Simplified protein models

- The **hydrophobic-polar (HP)** model
  - The simplest - yet non-trivial - abstraction for the protein structure prediction problem
- A protein structure with  $n$  amino acids  $\Leftrightarrow$  a sequence  $S = s_1 \dots s_n$ ,  $s_i, i=1 \dots n$  can be either  $H$  or  $P$

A valid protein configuration for a self-avoiding path on a regular lattice with vertices labeled by amino acid.



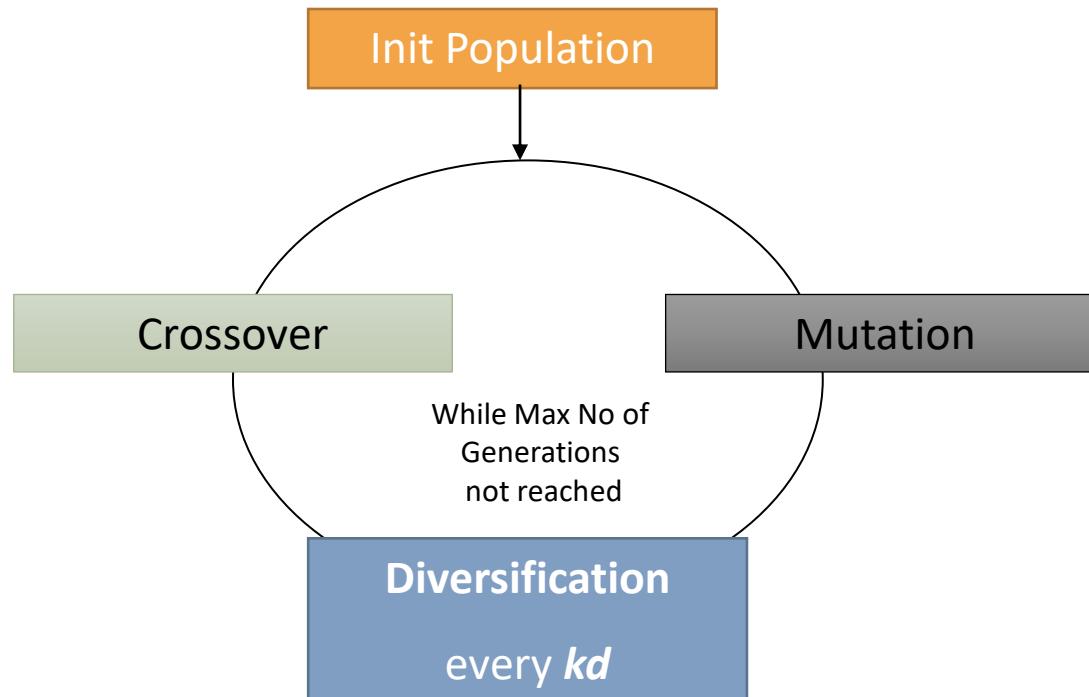
$$S = HHHPHPPPPPH$$

# Protein Structure Prediction

- Elements of a given pair of residues are considered topological neighbors if they are adjacent in the lattice and not consecutive in the sequence
- The energy associated to a protein conformation takes into account every pair of H residues which are topological neighbors
- Every H-H topological contact contributes -1 to the energy function
- **Objective:** find the protein conformation with minimum energy

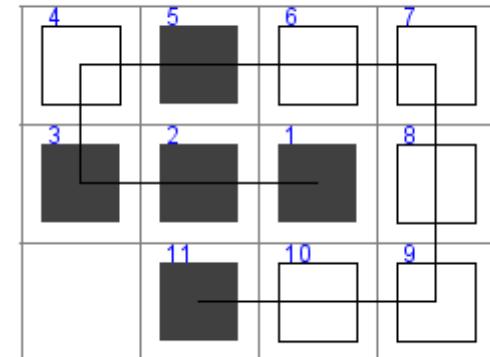
# Evolutionary Model with Hill-Climbing Search Operators

- The population size is fixed and offspring are asynchronously inserted in the population



# Representation

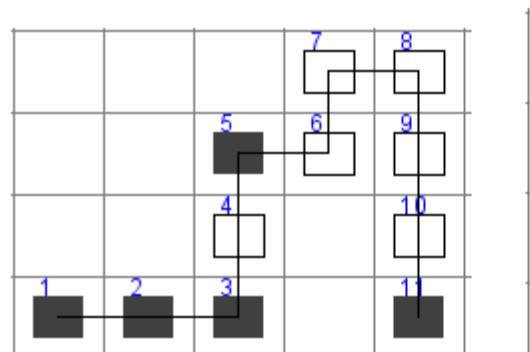
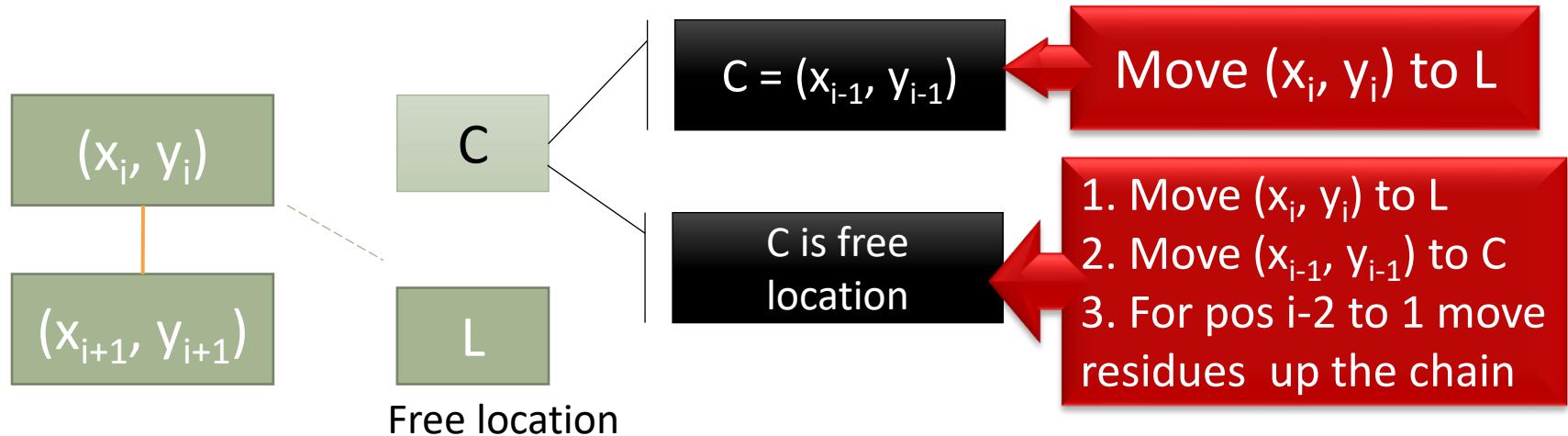
- Internal coordinates representation
- For a protein HP sequence with  $n$  residues  $S = s_1 \dots s_n$ , the chromosome length is  $n-1$ , each position in the chromosome encodes the direction
  - L(Left)
  - U(Up)
  - R(Right)
  - D(Down)towards the location of the current residue relative to the previous one.



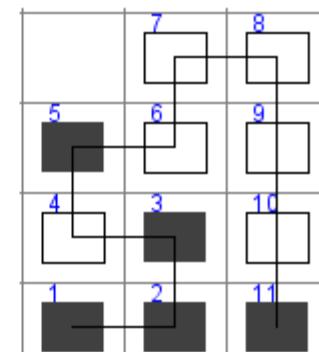
LLURRRRDDLL

# Mutation: Pull Moves (Lesh, 2003)

- A single residue is moved diagonally causing the transition of connecting residues



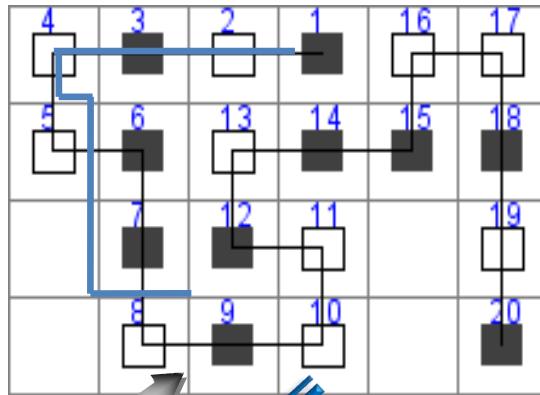
Carreira et al.



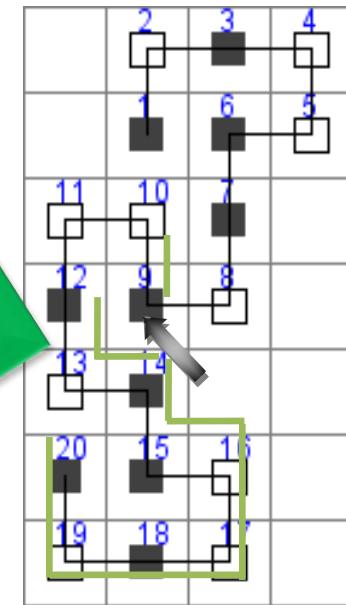
RULURURDDD

# Crossover Operator

First Parent

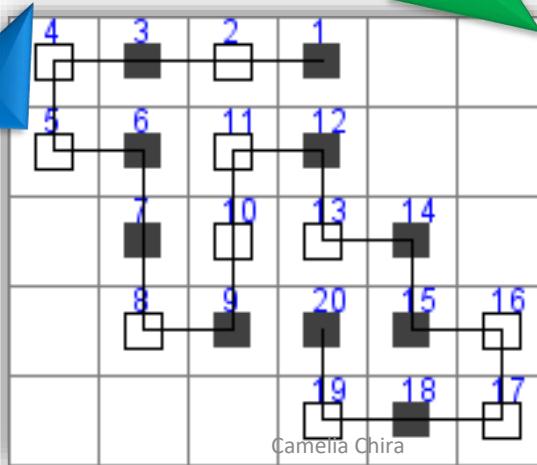


Second Parent



Positions 1 to 9

Positions after 9 if  
valid



# Diversification

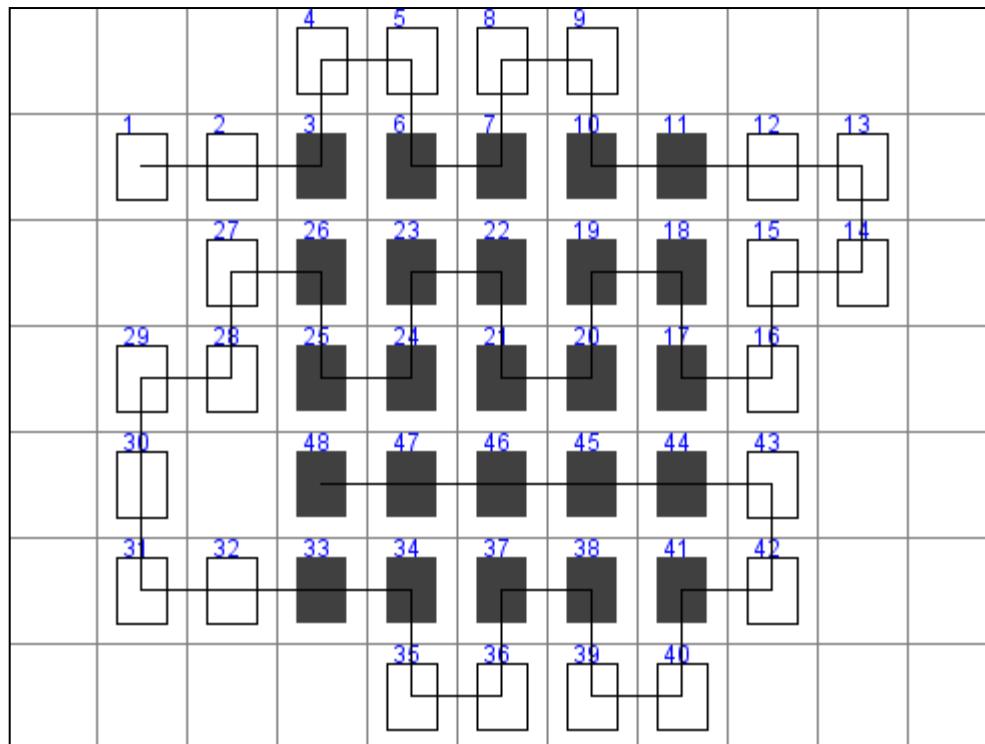
- Individuals from the current population are grouped based on their fitness (one group for each fitness value)
- For each group identified, subgroups of similar individuals are constructed based on the Hamming distance
  - Individuals are considered similar if  $distH < (n-1)/4$
- For each subgroup of similar individuals, one of them is kept in the current population and the rest of individuals are replaced by new randomly generated chromosomes (improved by a hill-climbing mutation).

# Results for Seq. S5 size 48, E = -23

- *HP sequence:* 2P 1H 2P 2H 2P 2H 5P 10H 6P 2H 2P 2H 2P 1H 2P 5H

- *Chromosome:*

RRURDRURDRRRDLDLULDLULDLULDDRRRDRURDRURULLLL



# Routing Optimization



- **Vehicle Routing Problem (VRP)**
  - Capacitated VRP
  - Multiple Depots VRP: 2 SC
  - VRP with Pickup and Delivery

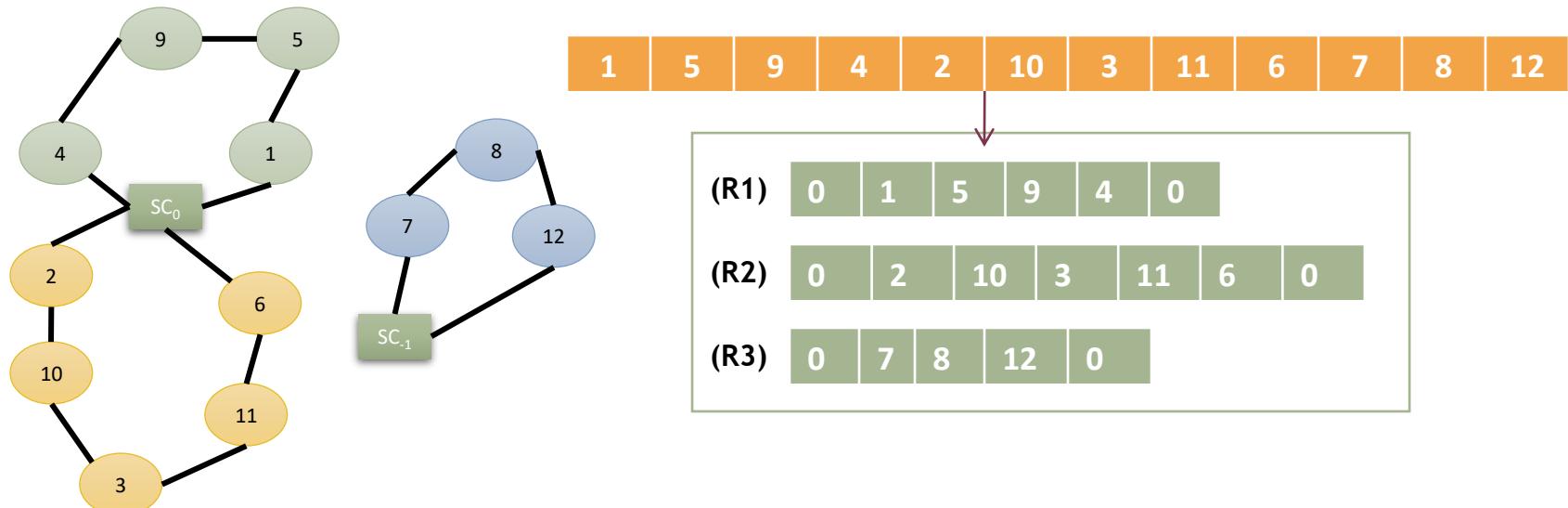
- Route for a vehicle  $k$ :  $R = \{i_1, i_2, \dots, i_{|R|}\}$ , where  $i_1 = i_{|R|} = -1$  or  $i_1 = i_{|R|} = 0$
- $R$  is feasible if:
  - ✓ The number of bicycles required by BS in the route is not higher than the capacity of the vehicle  $\sum_{p=2}^{|R|-1} (d_{i_p} - s_{i_p}) \leq Q_k$
  - ✓ For each BS: a positive demand should not exceed the current vehicle load  $y_k$  while a pickup demand should not exceed  $Q_k - y_k$

# EA Features

- Fitness function: total cost of routes
- Fixed size population initially randomly generated
- Elite subpopulation + roulette-wheel selection
- Order crossover
- Swap mutation
- Asynchronous replacement of first parent: occurs during the same generation and the newly generated offspring might be selected as a mate within the same generation

# Representation

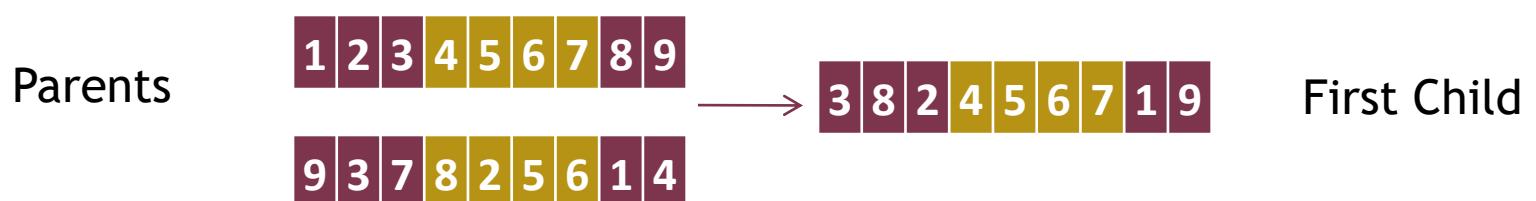
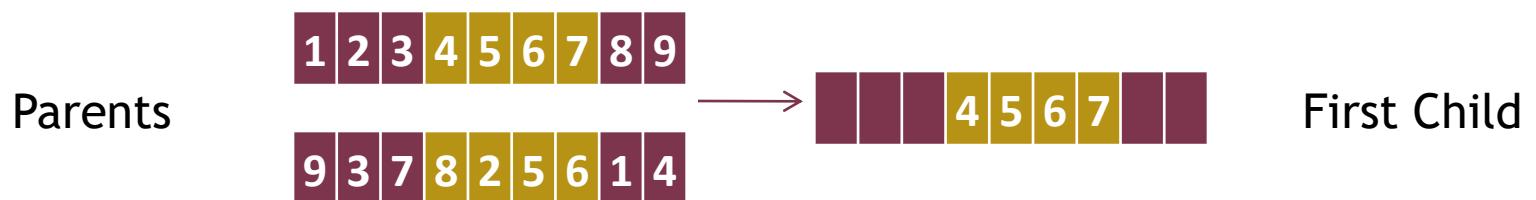
- An individual incorporates all the routes to be executed and is defined as a permutation of the node
- The SC (nodes -1 and 0) are not included in the representation as they will be automatically determined based on the nearest SC to the first node in the route



- Nodes are assigned to a route in the order they appear in the permutation
- *A new route is created when the current node in the permutation can not be serviced by the same vehicle*

# Order crossover

Order crossover is applied to this pair of parents with a certain probability.

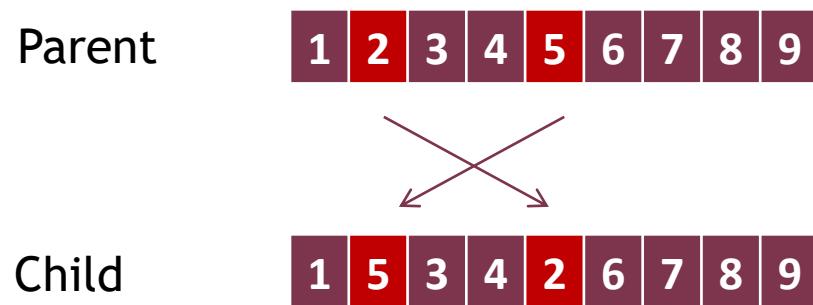


Second Child → 

3	4	7	8	2	5	6	9	1
---	---	---	---	---	---	---	---	---

# Swap mutation

**Swap mutation** is applied to the current individual from the intermediary population with a certain probability.



# Recapitulare

- Probleme complexe
  - Spatiul de cautare, Marimea problemei
- Cautare locală
  - Functii de vecinatate, Algoritmi Hill-Climbing
- Algoritmi single-solution
  - Simulated Annealing, Tabu Search
- Algoritmi Evolutivi
  - Reprezentare, fitness, selectie, variație
  - Selectia si managementul populatiei, setarea parametrilor
- Algoritmi inspirati de natura (swarm intelligence)
  - PSO, ACO
- Invatare automata
- Modele hibride