Invatare automata

Universitatea Politehnica Bucuresti

Anul universitar 2021-2022

Adina Magda Florea

Curs Nr. 6

Reţele neurale cu auto-organizare

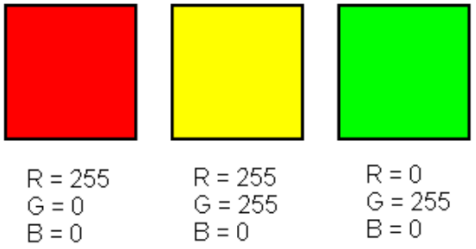
- **Retele Kohonen (SOM)**

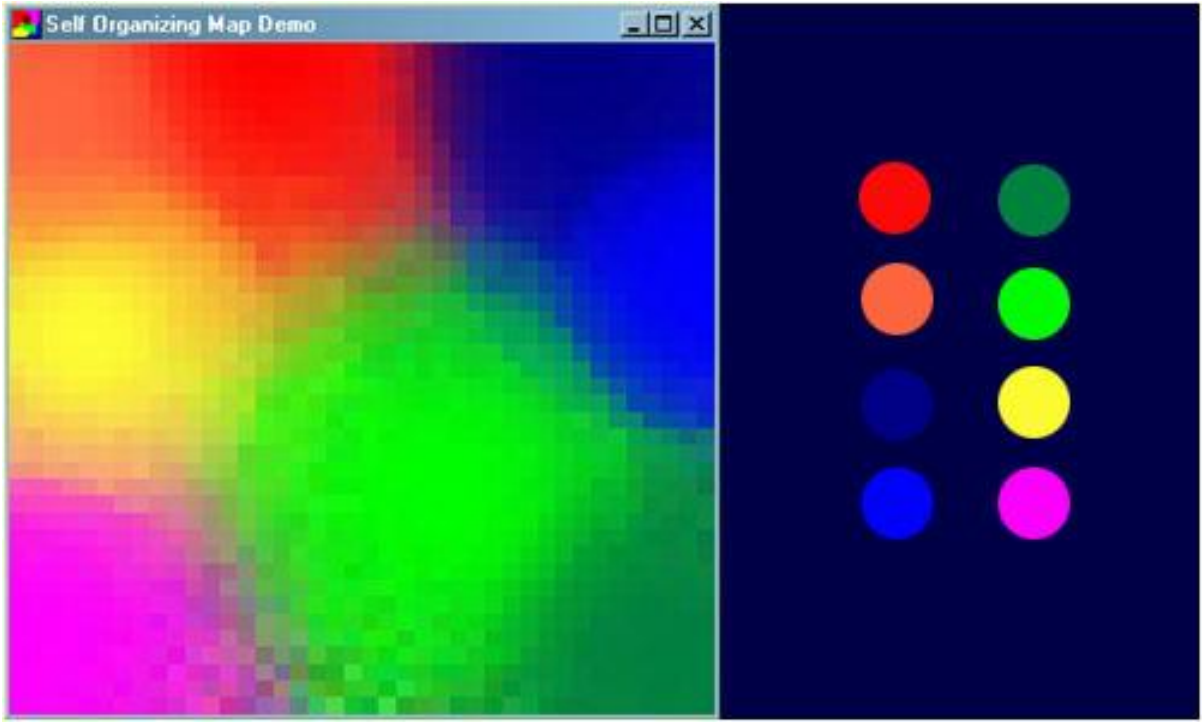
Rețele Kohonen

- Retele de tip harta cu auto-organizare a caracteristicilor (SOM - **self-organizing feature maps**) - inventate de Teuvo Kohonen, profesor la Academia din Finlanda, 1989
- Se inspira din modelul biologic – pozitia neuronilor are importanta
- ***Invatare nesupervizata***
- Ofera o modalitate de a reprezenta date multi-dimensionale in spatii dimensionale reduse, de obicei 1-2 dimensiuni = ***cuantificarea vectorilor***
- Tehnica Kohonen creaza o retea care memoreaza informatia astfel incat orice relatie topologica din setul de invatare este mentinuta.

EXEMPLU de utilizare a unei retele SOM

- Maparea culorilor pe baza componentelor RGB (3D) in doua dimensiuni – o retea SOM antrenata sa recunoasca 8 culori
- Culorile au fost prezentate retelei ca vectori 3D, cu cate o dimensiune pt. fiecare componenta a culorii – reteaua a invatat sa reprezinte culorile intr-un spatiu 2D.
- Pe langa clusterizarea culorilor in regiuni distincte, regiunile cu proprietati similare sunt plasate de obicei adiacent una de cealalta.



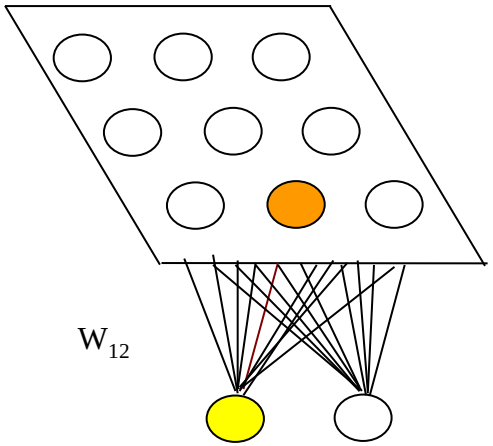


Idei SOM

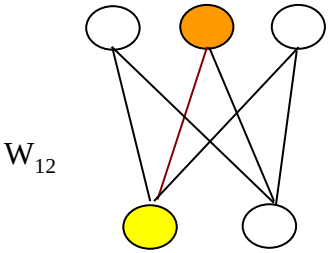
- La prezentarea exemplelor ponderile creaza clustere care impart spatiul de intrare a.i iesirile activate tind sa aproximeze functia de densitate de probabilitate a intrarii
- Ponderile sunt actualizate a.i. noduri apropiate topologic sunt senzitive la intrari fizic similare = ***topogy preserving maps***
- **M** noduri de iesire si **N** intrari
- In timpul invatarii, unitatea (dintr-un cluster) care are ponderi cat mai apropiate de valorile de intrare (distanta Euclidiana minima) este aleasa pe post de ***castigator***
- Unitatea castigatoare si vecinii ei isi actualizeaza ponderile
- Necesita definirea unei ***vecinatati*** in jurul fiecarei unitati, care descreste in timp

Structura SOM

- Iesirile retelei pot fi organizate unidimensional (vector) sau bi-dimensional (matrice)

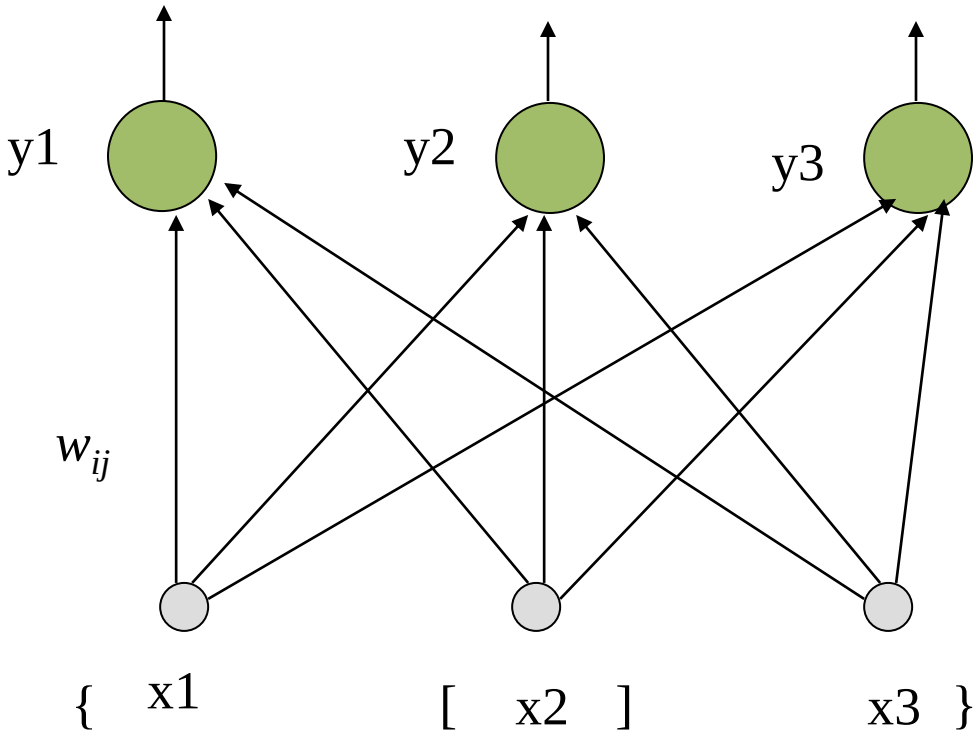


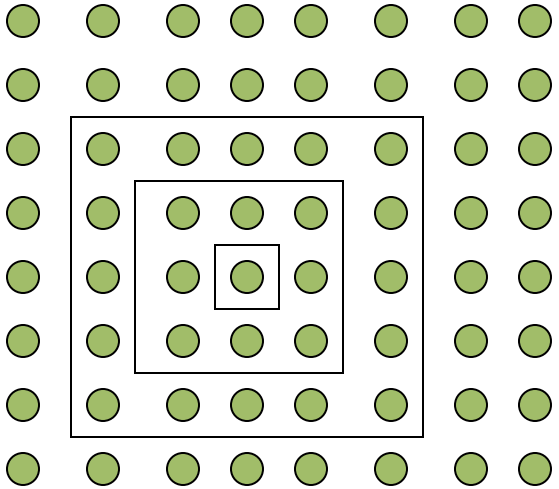
Retea cu 2 intrari (N=2) si 9 iesiri (M=9) (bi-dimensionala)



Retea cu 2 intrari (N=2) si 3 iesiri (M=3) (uni-dimensionala)

Structura SOM





R = 0
R = 1
R = 2

Algoritmul de invatare SOM

1. Initializeaza ponderile, fixeaza dim. vecinatatii topologice **R** si rata de invatare $\alpha(t_0)$

   (initial ponderi aleatore cu val mici sau nealeatoare)

2. **pentru** fiecare x^s_i **executa**

   2.1 calculeaza distanta pt toate nodurile de iesire

   d_j – dist intre intrari **i**(i=1,N) si noduri iesire **j**(j=1,M)

   $x^s_i(t)$ – intrarea **i** la momentul **t**

   $w_{ij}(t)$ – ponderea intre intrarea **i** si nodul de iesire **j**

   $$d_j = \Sigma_{i=1,N}(x^s_i(t) - w_{ij}(t))^2$$

   2.2 Selecteaza nodul cu distanta minima: **j* cu d_j min**

   2.3 Actualizeaza ponderile unitatii **j*** si ale unitatilor din vecinatatea Ne_{j^*}

   $$w_{ij}(t+1) = w_{ij}(t) + \alpha(t)\,(x^s_i(t) - w_{ij}(t))$$

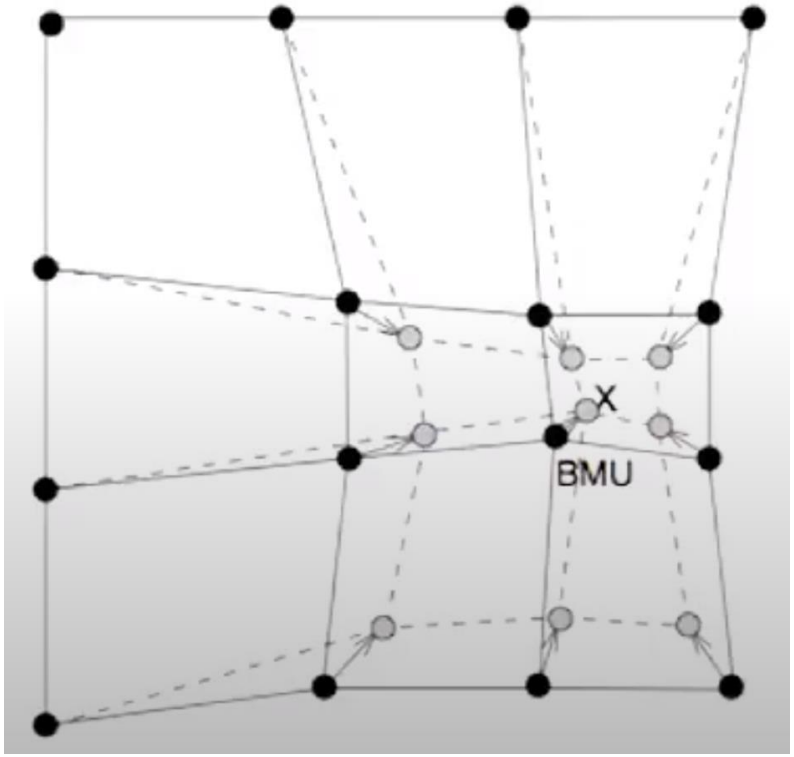
   pentru **j** in NE_{j^*} la momentul **t**, i=1,N

3. Reduce **R**

4. **repeta** de la 2 pana conditie de oprire

sfarsit

j* - Best Matching Unit (BMU)
La fiecare iteratie BMU si vecinii sunt mutati mai aproape de vectorul de intrare



Exemplu

- 4 vectori (s=4) cu 4 elemente (N=4)
  (1 1 0 0) (0 0 0 1) (1 0 0 0) (0 0 1 1)
- 2 clustere (M=2)
- $\alpha(0) = 0.6$ $\alpha(t+1)=0.5*\alpha(t)$
- Ponderi initiale

  .2 .8
  .6 .4
  .5 .7
  .9 .3

  R=0 $d_j = \Sigma_{i=1,N}(x^s_i(t) - w_{ij}(t))^2$

  (1,1,0,0) $D(1) = (0.2-1)^2+(0.6-1)^2+(0.5-0)^2+(0.9-0)^2=1.86$
  D(2) $=(0.8-1)^2+(0.4-1)^2+0.7-0)^2+(0.3-0)^2 =$ **0.98**
  J=2

  $w_{i2}^{nou}=w_{i2} +0.6(x_i-w_{i2})$

  .2 .92
  .6 .76
  .5 .28
  .9 .12

$$w_{ij}(t+1) = w_{ij}(t)+ \alpha(t) (x^s_i(t) - w_{ij}(t))$$

- (0 0 0 1)
  .08 .92
  .24 .76
  .20 .28
  .96 .12
- (1 0 0 0)
  .08 .968
  .24 .304
  .20 .112
  .96 .048
- (0 0 1 1)
  .032 .968
  .096 .304
  .680 .112
  .984 .048

- Se reduce rata de invatare
- α=0.5*0.6 = 0.3
- Epoca 100 matricea devine
  6.7e-17 1.0000
  2.0e-16 0.4900
  0.5100 2.3e-16
  1.0000 1.0e-16

  0.0 1.0
  0.0 0.5
  0.5 0.0
  1.0 0.0

(1 1 0 0) (0 0 0 1) (1 0 0 0) (0 0 1 1)

Cati neuroni (ponderi) se actualizeaza

WTA (Winner Takes All) –

selecteaza neuronul cu distanta Euclidiana minima (castigator)

si actualizeaza numai ponderile acestuia

- **WTM** (Winner Takes Most) – selecteaza neuronul cu distanta Euclidiana minima si actualizeaza ponderile acestuia si a neuronilor dintr-o vecinatate de raza R
- WTM are o convergenta mai buna decat WTA
- In WTM mai multi neuroni au ponderile actualizate in fiecare iteratie.

$$w_{ij}(t+1) = w_{ij}(t) + \alpha(t)\,(x^s_i(t) - w_{ij}(t))$$

pentru i=1,N, j=j* si j $\in NE_{j^*}$ de raza R(t),

s =1,Nr exemple

• Viteza de invatare si raza vecinatatii se modifica la fiecare epoca conform:

$$\alpha(t+1) = \Delta\alpha\ \alpha(t) \qquad R(t+1) = \Delta R\ R(t)$$

cu $\Delta\alpha$ si ΔR factori constanti

De ex $R(t) = 0.5s \exp(-t\log(0.5s)/t)$ cu s nr de puncte ale laturii hartii

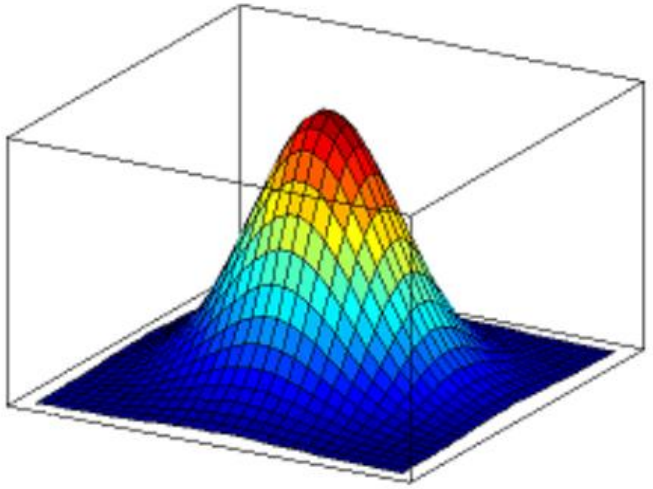
• Actualizare ponderi – varianta

$$\mathbf{w_{ij}(t+1) = w_{ij}(t) + \alpha(t)\ \beta(t)(x^s_i(t) - w_{ij}(t))}$$

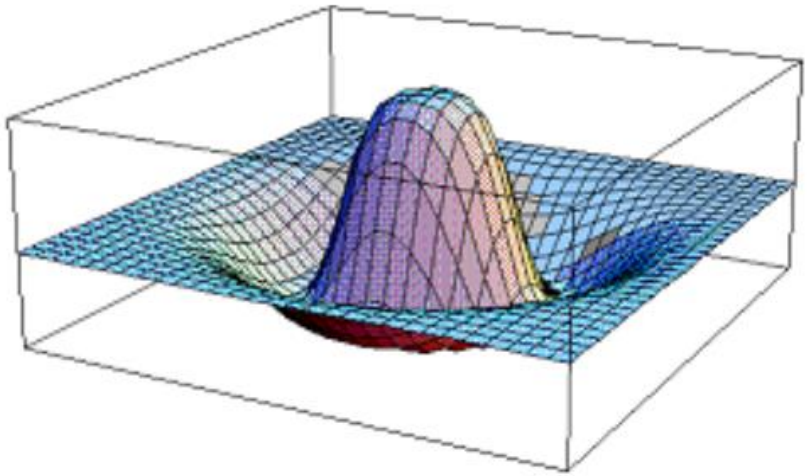
$$\mathbf{\beta(t) = \exp(-\ d_j / R(t)^2)}$$

Alte modele de vecinatate

Distributie gausiana



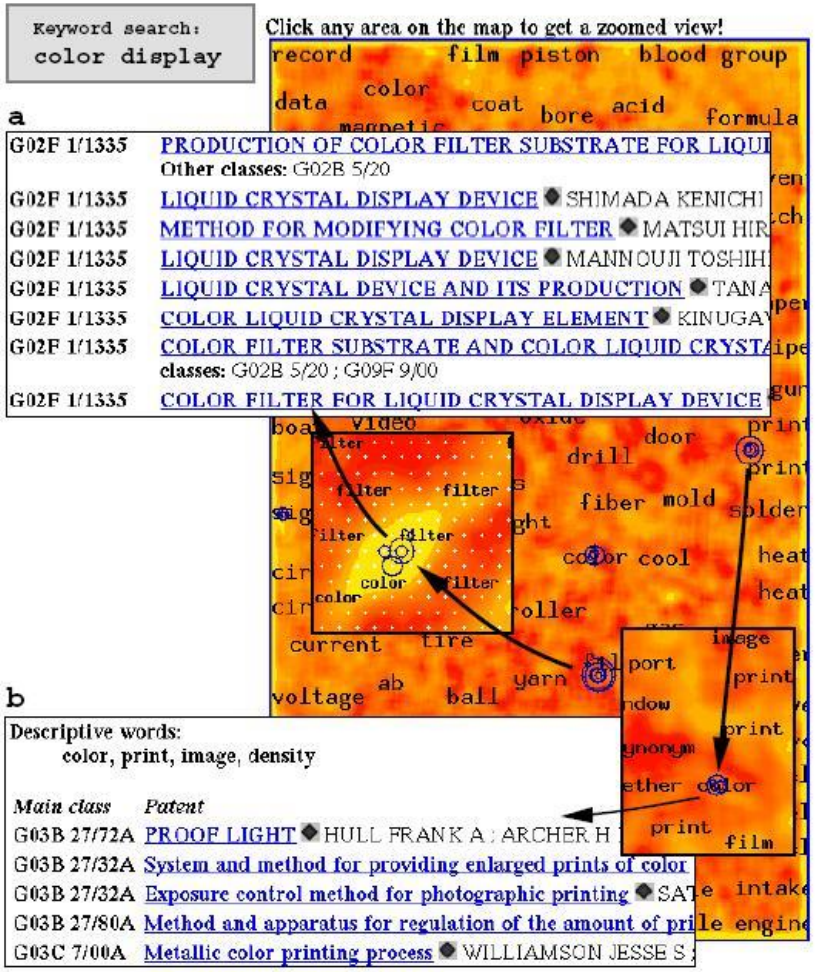
Palarie Mexicana
(Maxican hat)



Exemplu: Proiectul WEBSOM

Prof Kohonen Univ Helsinki

- Organizarea si vizualizare colectiilor de documente in timp real cu un SOM
- Dimensiunea intrarii –43,222
- 1 dimensiune corespunde frecventei de aparitie a unui cuvant in document
- SOM cu 1,002,240 neuroni
- 6,840,568 documente cu patente
- **Optimizari pt cresterea vitezei de invatare**



- **Optimizare: Reducerea dimensiunii**

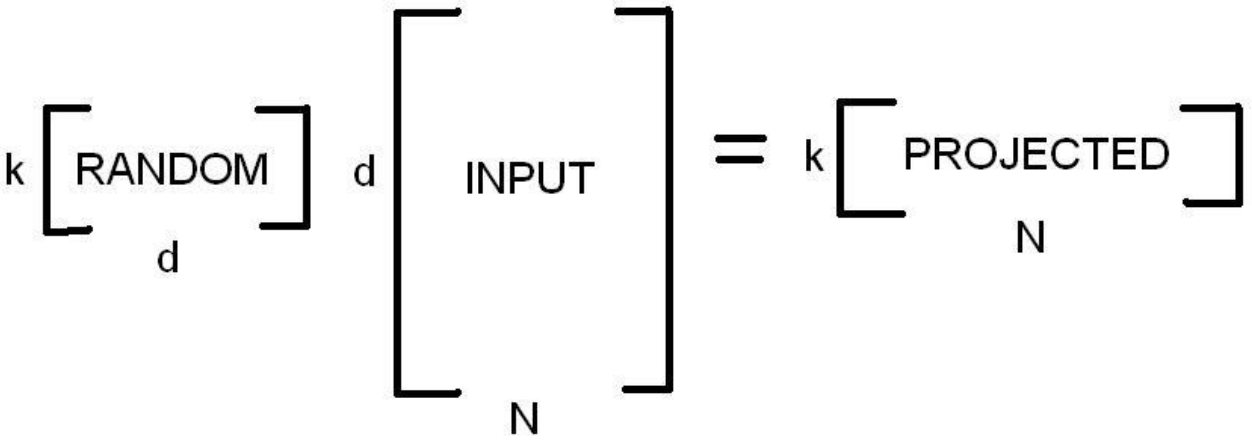
Algoritmul Random Projection

Vectori histograma

- fiecare componenta a vectorului corespunde frecventei de aparitie a unui cuvant
- Cuvintele sunt associate cu ponderi care reflecta puterea lor de discriminare intre topici
- De gasit un subset adecvat de vectori care grupeaza bine documentele
  - **Randomly Projected Histograms**
- Pornind de la dimensiunea d originala a datelor X se proiecteaza intr-un subspatiu de dimensiune k (k << d)

- Foloseste o matrice aleatoare R de dimensiune k x d, elementele din fiecare coloana sunt vectori cu valori cu o distributie normala de lungime unitara

  $R_{k \times d}\ X_{d \times N}$ => noua matrice $X_{k \times N}$



De ce functioneaza?

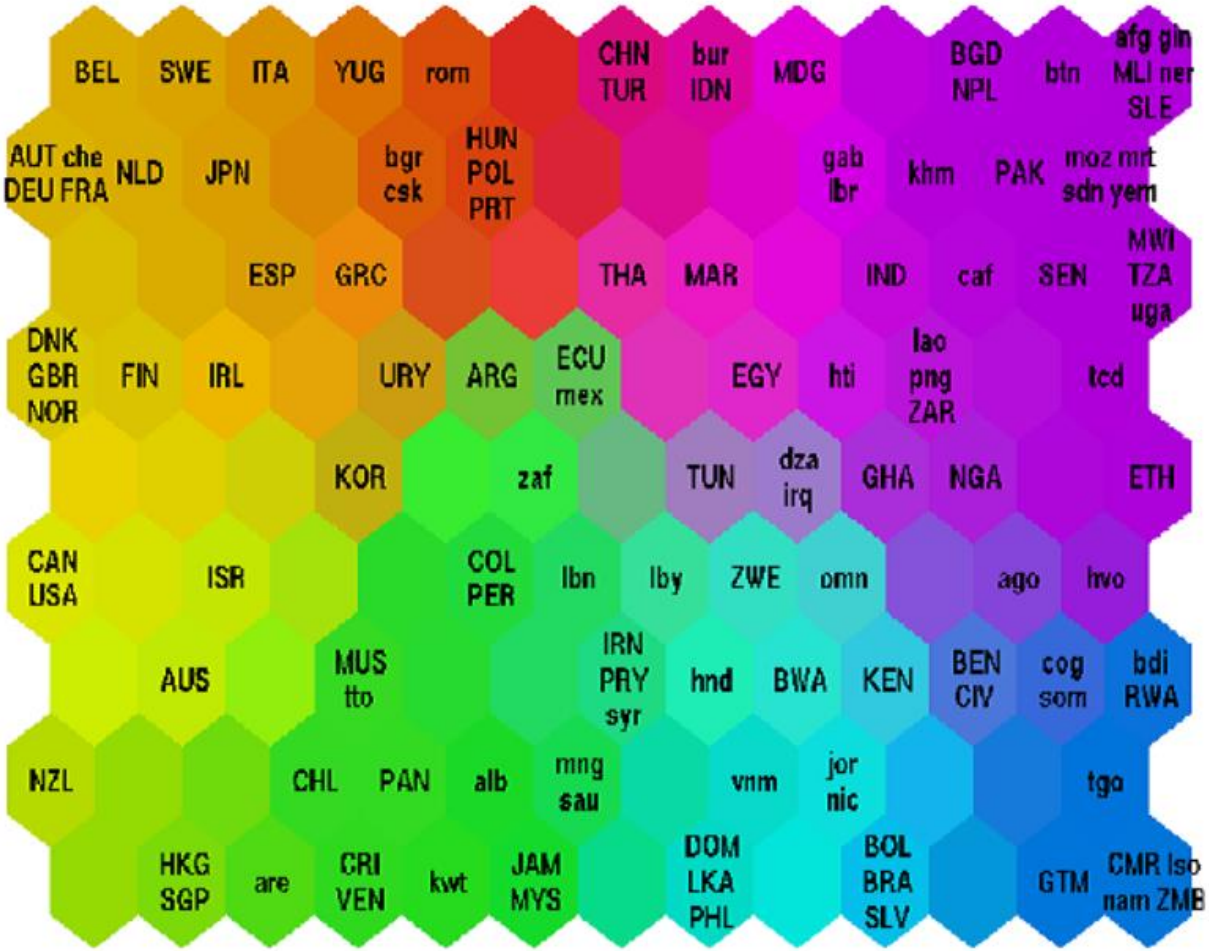
- Lema Johnson – Lindenstrauss

  “Daca punctele dintr-un spatiu de n dimensiuni sunt proiectate intr-un spatiu ales aleator de o dimensiune mai mica convenabila, atunci distanta dintre puncte este aproximativ pastrata”
- Similaritatea unei perechi de vectori proiectati este aceeasi, in medie, cu similaritatea perechii corespunzatoare de vectori originali
- Similaritatea este calculate ca produs scalar al celor 2 vectori

- **Alte 2 optimizari**
- Ponderi pregenerate
- Limitarea cautarii neuronilor castigatori

Alt proiect care a utilizat SOM

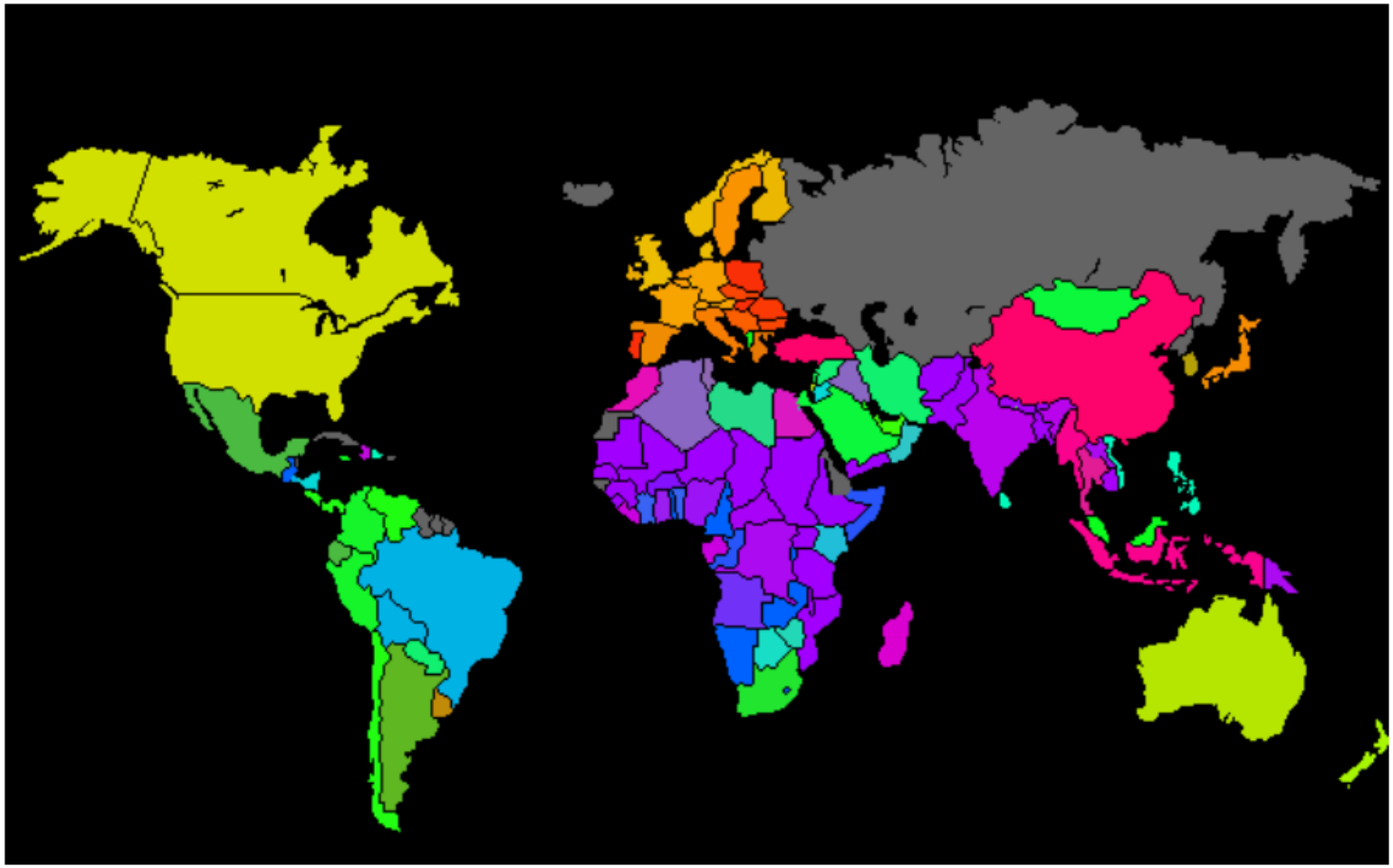
- Classifying World Poverty



The Country Names

AFG	Afghanistan	GTM	Guatemala	NZL	New Zealand
AGO	Angola	HKG	Hong Kong	OAN	Taiwan, China
ALB	Albania	HND	Honduras	OMN	Oman
ARE	United Arab Emirates	HTI	Haiti	PAK	Pakistan
ARG	Argentina	HUN	Hungary	PAN	Panama
AUS	Australia	IDN	Dutch East Indies	PER	Peru
AUT	Austria	IDN	Indonesia	PHL	Philippines
BDI	Burundi	IND	India	PNG	Papua New Guinea
BEL	Belgium	IRL	Ireland	POL	Poland
BEN	Benin	IRN	Iran, Islamic Rep.	PRT	Portugal
BGD	Bangladesh	IRQ	Iraq	PRY	Paraguay
BGR	Bulgaria	ISR	Israel	ROM	Romania
BOL	Bolivia	ITA	Italy	RWA	Rwanda
BRA	Brazil	JAM	Jamaica	SAU	Saudi Arabia
BTN	Bhutan	JOR	Jordan	SDN	Sudan
BUR	Myanmar	JPN	Japan	SEN	Senegal
BWA	Botswana	KEN	Kenya	SGP	Singapore
CAF	Central African Rep.	KHM	Cambodia	SLE	Sierra Leone
CAN	Canada	KOR	Korea, Rep.	SLV	El Salvador
CHE	Switzerland	KWT	Kuwait	SOM	Somalia
CHL	Chile	LAO	Lao PDR	SWE	Sweden
CHN	China	LBN	Lebanon	SYR	Syrian Arab Rep.
CIV	Cote d'Ivoire	LBR	Liberia	TCD	Chad
CMR	Cameroon	LBY	Libya	TGO	Togo
COG	Congo	LKA	Sri Lanka	THA	Thailand
COL	Colombia	LSO	Lesotho	TTO	Trinidad and Tobago
CRI	Costa Rica	MAR	Morocco	TUN	Tunisia
CSK	Czechoslovakia	MDG	Madagascar	TUR	Turkey
DEU	Germany	MEX	Mexico	TZA	Tanzania
DNK	Denmark	MLI	Mali	UGA	Uganda
DOM	Dominican Rep.	MNG	Mongolia	URY	Uruguay
DZA	Algeria	MOZ	Mozambique	USA	United States
ECU	Ecuador	MRT	Mauritania	VEN	Venezuela
EGY	Egypt, Arab Rep.	MUS	Mauritius	VNM	Viet Nam
ESP	Spain	MWI	Malawi	YEM	Yemen, Rep.
ETH	Ethiopia	MYS	Malaysia	YUG	Yugoslavia
FIN	Finland	NAM	Namibia	ZAF	South Africa
FRA	France	NER	Niger	ZAR	Zaire
GAB	Gabon	NGA	Nigeria	ZMB	Zambia
GBR	United Kingdom	NIC	Nicaragua	ZWE	Zimbabwe
GHA	Ghana	NLD	Netherlands		
GIN	Guinea	NOR	Norway		
GRC	Greece	NPL	Nepal		

- Classifying World Poverty
- World Poverty Classifier - SOM care mapeaza tarile pe baza a 39 de factori care cuantifica calitatea vietii, de ex stare de sanitate, calitatea nutritiei, educatie, etc.
- Colorarea tarilor pe harta lumii pe baza clasificarii anterioare



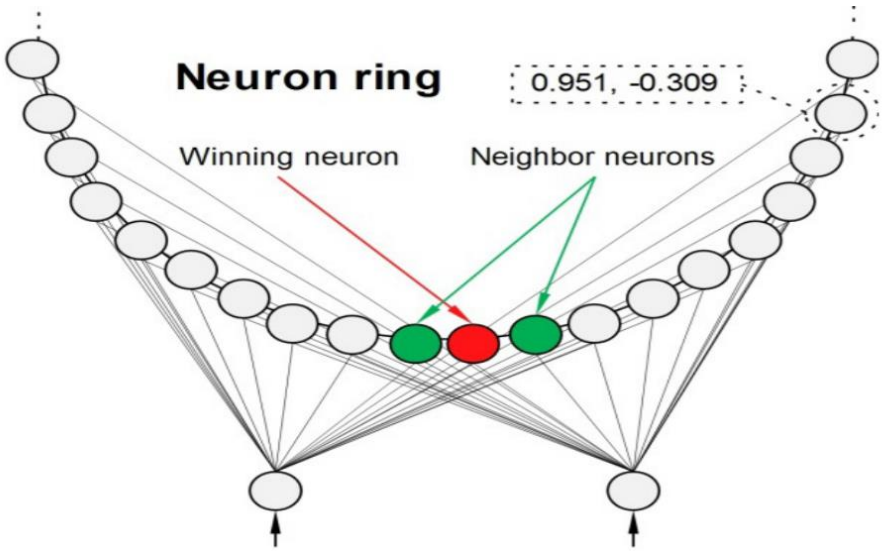
Exemplu: Problema comis voiajorului

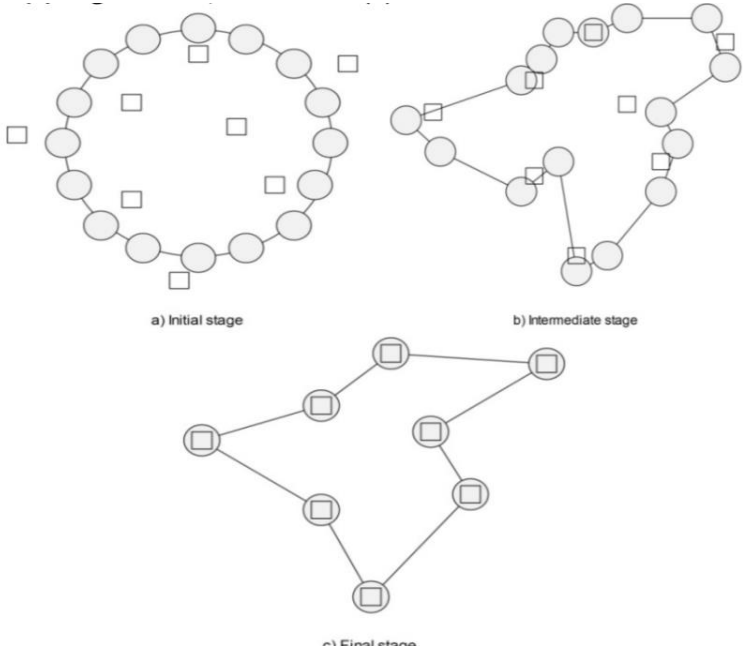
- Coordonatele orasului – intrari
- Unitati cluster = nr de orase
- Topologie liniara circulara

- Se utilizeaza o retea Kohonen uni-dimensionala
- Numarul de neuroni este egal cu numarul de orase
- Daca ponderile unui neuron sunt egale cu coordonatele unui oras atunci acest neuron reprezinta orasul
- Neuronii sunt organizati intr-un vector. Acest vector reprezinta secventa de orase vizitate.
- DAR valorile ponderilor unui neuron nu sunt de obicei egale (exact) cu coordonatele orasului desi ponderile sunt aproriate de valorile coordonatelor oraselor.

Problema comis voiajorului

- Este necesara utilizarea unui algoritm de "**reparare**" care modifica ponderile a.i. sa ajunga egale cu coordonatele oraselor. In algoritm, daca un neuron $\mathbf{n_i}$ este atribuit unui oras **A** aceasta inseamna ca ponderile neuronului $\mathbf{n_i}$ sunt egale cu coordonatele orasului **A**.





Algoritm de reparare

1. **pentru** fiecare neuron *j* = 1,M **executa**
   1.1 *Oras_apropiat* = cel mai apropiat oras de neuronul *j*
   1.2 **daca** *Oras_apropiat* nu este atribuit unui neuron **atunci**
      - atribuie *Oras_apropiat* neuronului *j*
      altfel elimina neuronul *j*
2. **pentru** fiecare oras *k* = 1,M **executa**
   2.1 **daca** orasul *k* nu este atribuit unui neuron **atunci**
      - creaza un nou neuron *t* si atribuie orasul *k* acestui neuron
      - *Neuron_apropiat* = cel mai apropiat oras/neuron de orasul *k*
      - insereaza noul neuron *t* inainte sau dupa *Neuron_apropiat*, in functie de care circuit este mai scurt

sfarsit

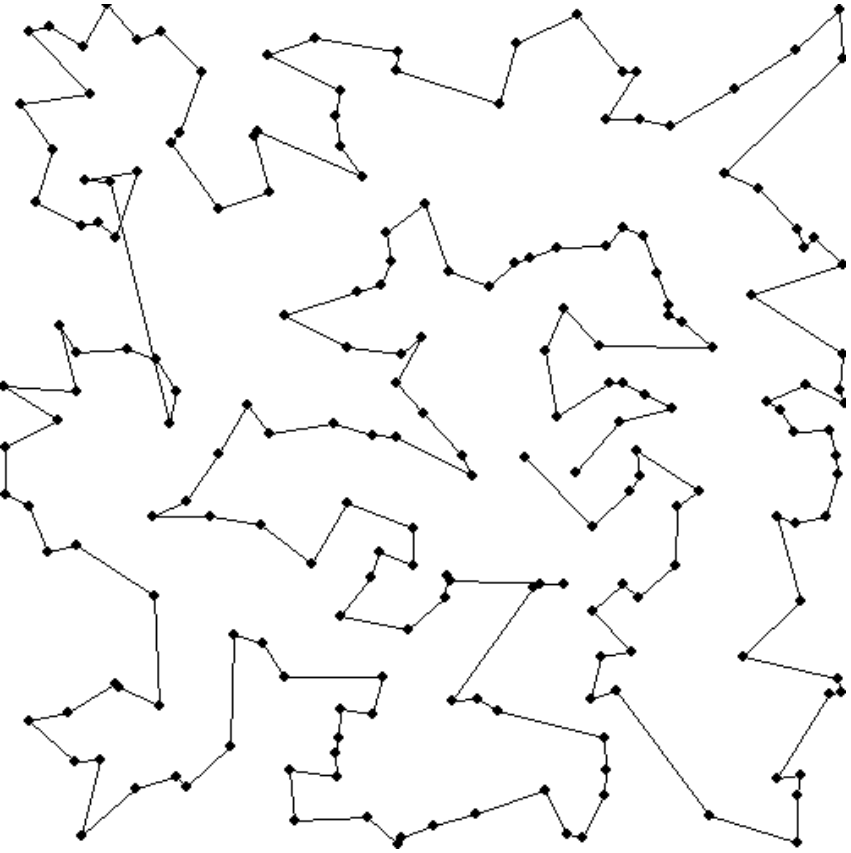
Problema comis voiajorului

- Dupa aplicarea acestui algoritm se gaseste destul de repede o solutie destul de buna (nu optima) dar care poate fi imbunatatita.
- Solutia poate fi in continuare optimizata folosind algoritmul *2opt*. De obicei *2opt* nu schimba cu mult solutia deoarece solutia este deja destul de buna.
- Algoritmul *2opt* selecteaza o parte a circuitului, il inverseaza, si il introduce inapoi in circuit. Daca noul circuit obtinut este mai scurt decat cel initial, atunci inlocuirea devine permanenta.
- De exemplu, daca avem un circuit (A, B, C, D, E, F) si o cale (B, C, D) care o inversam, atunci noul circuit este (A, D, C, B, E, F).
- Algoritmul se opreste daca nu se mai obtin imbunatatiri.

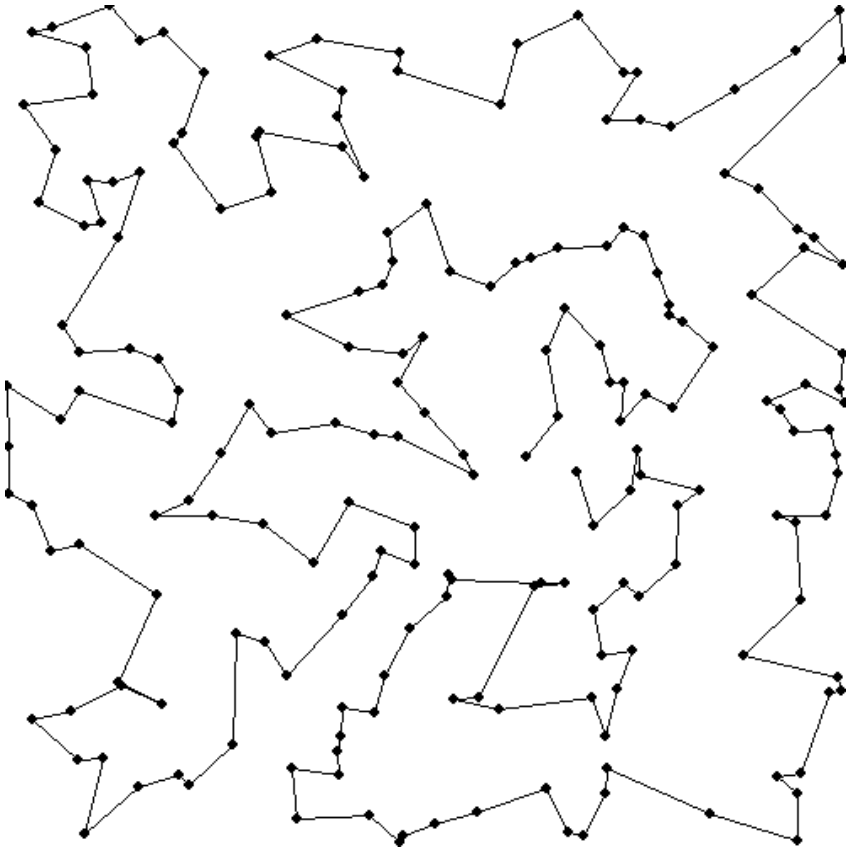
Problema comis voiajorului

- Valori experimentale (bune) gasite pentru modificarea vitezei de invatare si a razei
- **100 orase $\alpha(0) = 0.6$, $\Delta\alpha = 0.9997$, $\Delta R = 0.999$**
- **500 orase $\alpha(0) = 0.7$, $\Delta\alpha = 0.999985$, $\Delta R = 0.9994$**
- **1000 orase $\alpha(0) = 0.69$, $\Delta\alpha = 0.99992$, $\Delta R = 0.9996$**
- In fiecare caz numarul de epoci este 25000.
- **Rezultate obtinute**
- S-au utilizat in teste atat orase generate aleator cat si seturi de date de test din TSPLIB

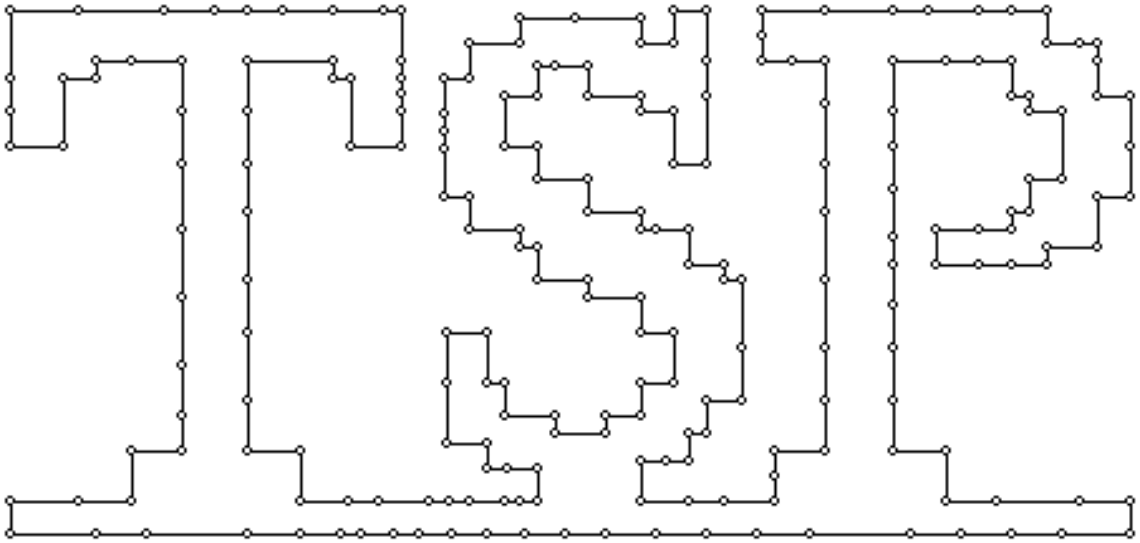
http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/



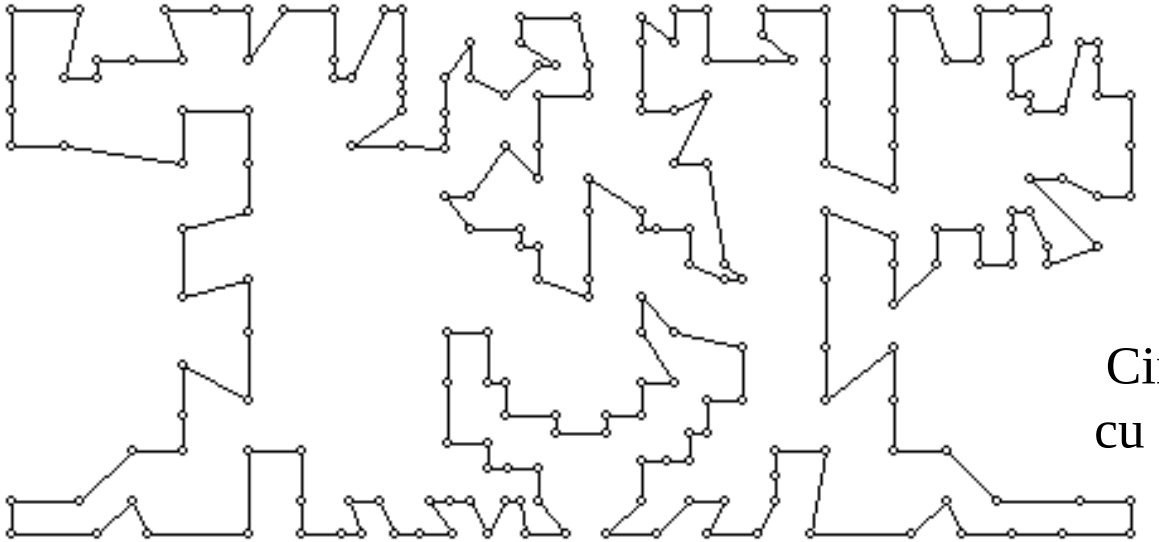
Solutie SOM fara optimizare *2opt*.
Exista 2 buce locale. Primul si ultimul neuron se pot vedea la mijlocul figurii.
Ei nu sunt conectati in figura dar distanta dintre ei este calculata.



Solutia SOM
dupa aplicarea optimizarii *2opt*



Circuit optim pentru
255 de orase,
obtinut din TSPLIB,
cu cost (optim) **3916**

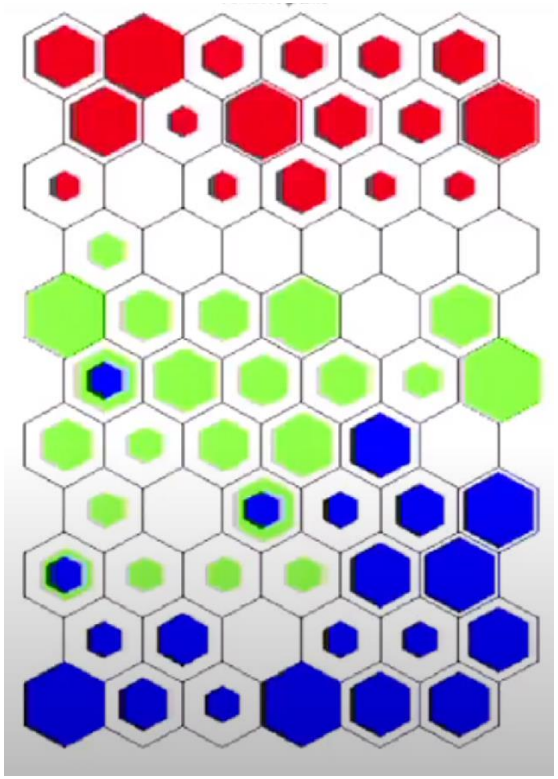


Circuit generat de SOM *2opt*,
cu cost **4130** (5.19% mai mare)

Vizualizare - Hit histogram (Hit maps)

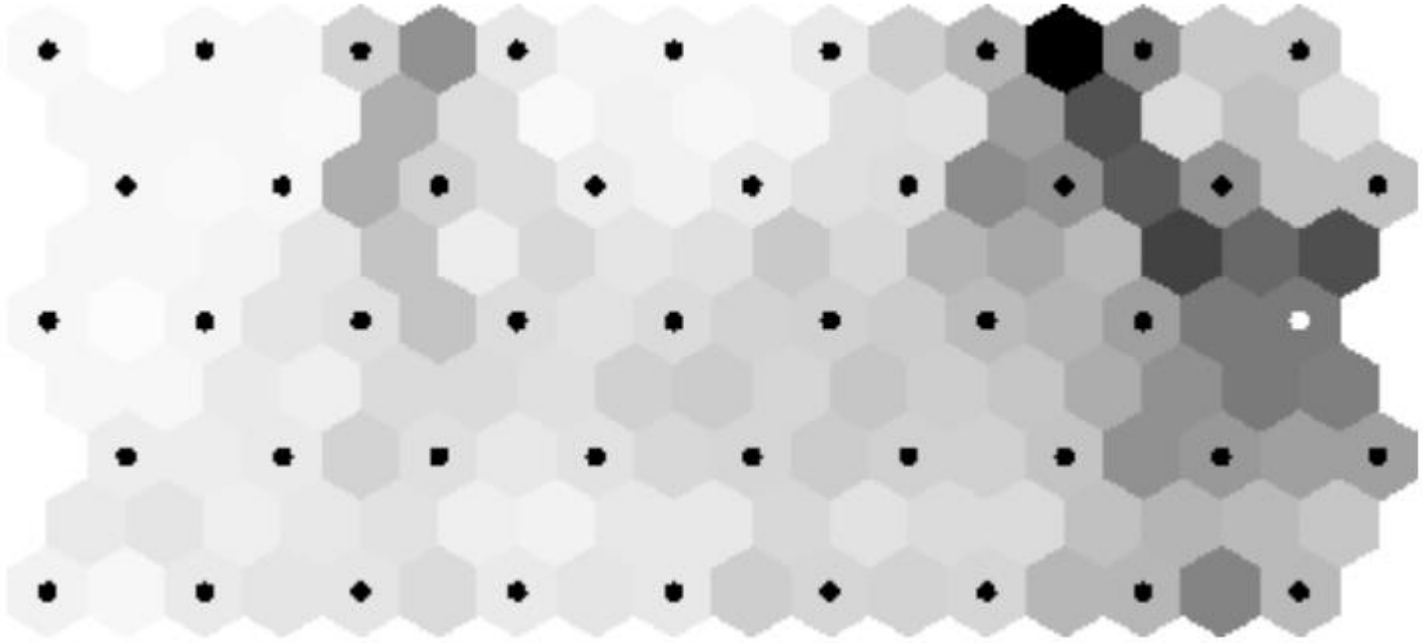
Arata distributia pe harta, de cate ori un neuron este ales ca BMU

Se formeaza luand un set de date, se afla BMU pt fiecare data din harta si crestem un contor asociat hartii de fiecare data cand se detecteaza un BMU



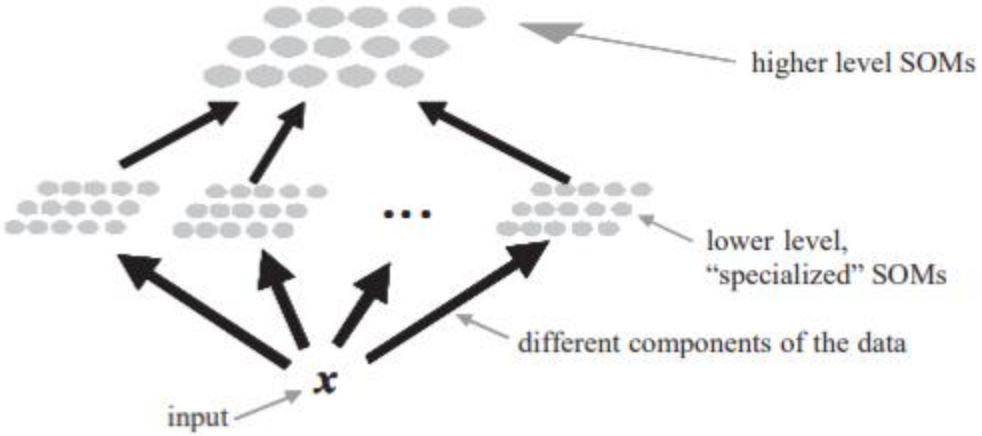
Vizualizare – U-matrix

Vizualizeaza distanta dintre neuroni. Distanta intre neuroni adiacenti este calculate si prezentata sub forma de nuante de gri sau diferite culori. O culoare inchisa intre neuroni corespunde unei distante mari si deci un gap intre ponderi si spatial de intrare. O culoare deschisa intre neuroni inseamna ca vectorii sunt apropiati unul de altul. Zonele deschise pot fi vazute ca grupuri (clusters) si cele inchise ca separatori ai grupurilor.



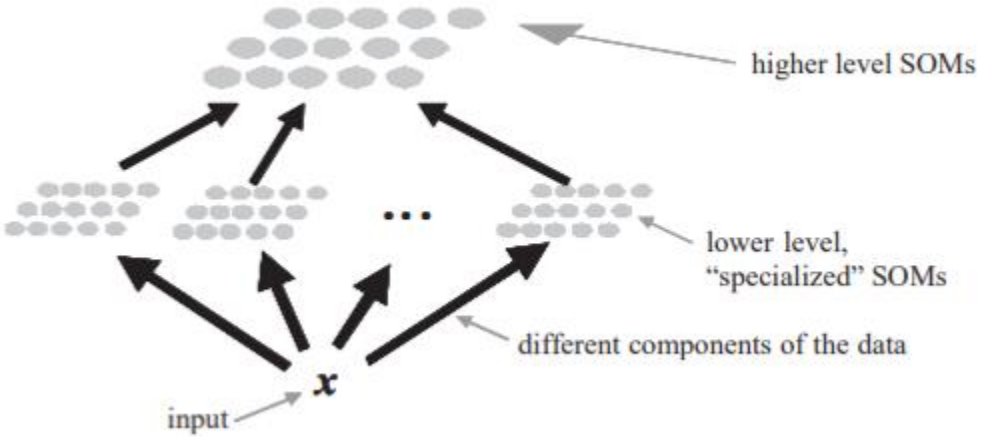
SOM Ierarhice

- Se utilizeaza in domenii in care o descompunere structurala in probleme mai mici stratificate este convenabila
- Una sau mai multe SOM sunt localizate pe nivele, fiecare harta operand pe diferite variabile tematice
- Folosite mult la intelegerea vorbirii, unde fiecare nivel trateaza diferite unitati de vorbire: foneme, silabe, parti de cuvant, cuvinte

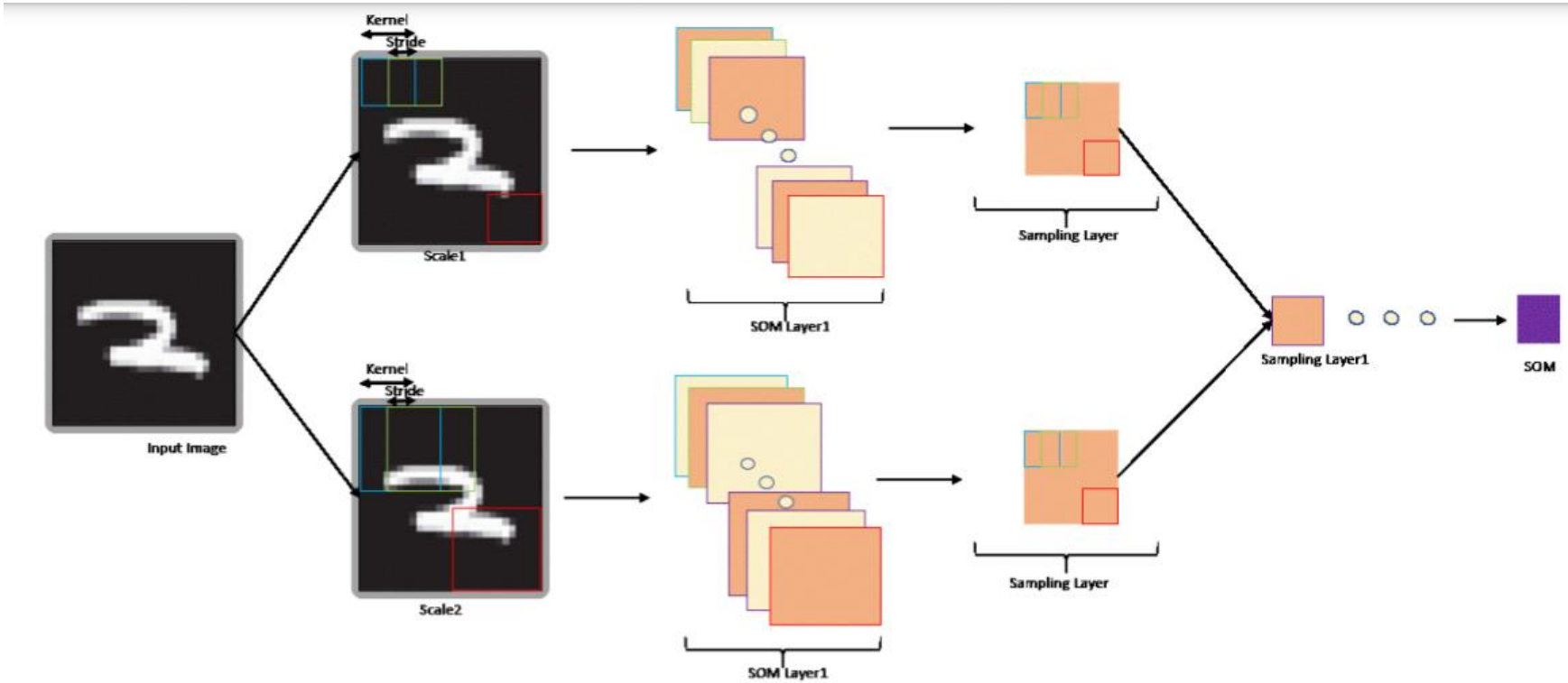


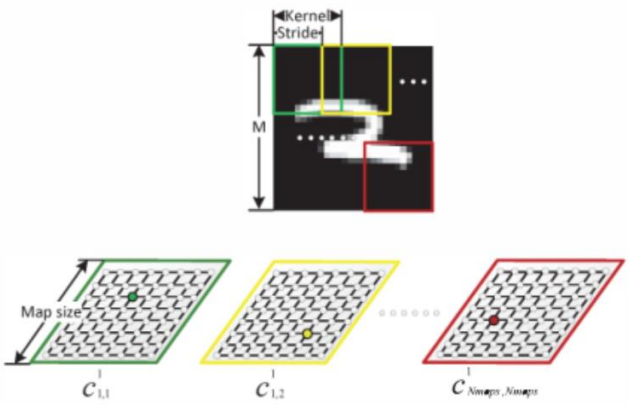
SOM Ierarhice

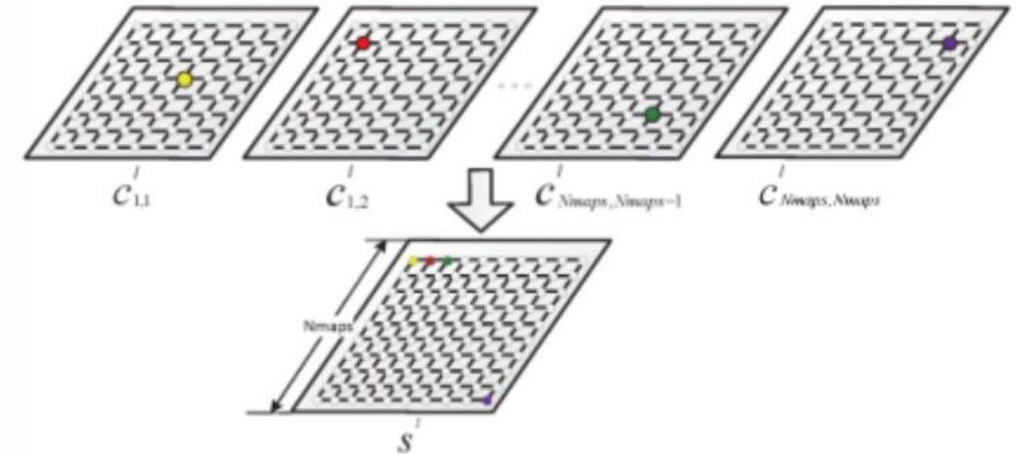
- Pot fi organizate cu diferite harti partiale la nivele inferioare care grupeaza datele in functie de diferite caracteristici si apoi trimit rezultatul catre SOM de pe nivele superioare
- Alternativ pot avea un SOM global pe nivelul inferior care functioneaza ca un mechanism de selectie pentru activarea uneia sau a mai multor harti care se specializeaza in anumite zone din spatial de intrare



Deep SOM


Kernel
Stride
Scale1
Input Image
Kernel
Stride
Scale2
SOM Layer1
SOM Layer1
Sampling Layer
Sampling Layer
Sampling Layer1
SOM


Kernel
Stride
M
Map size


Nmaps

Invatare automata

Universitatea Politehnica Bucuresti

Anul universitar 2021-2022

Adina Magda Florea

Curs nr. 7

- Invatarea prin recompensa - Intro
- Sistem Markov cu recompensa
- Proces de decizie Markov (MDP)
- Reinforcement learning (RL)
- Exemplu simplificat de RL
- Generalizare RL

1. RL - Introducere

- Mult timp – probleme simple
- Ultimii 5-10 ani – multe aplicatii diferite
- Atari game – David Silver 2015
- Planificare, Roboti, Optimizare

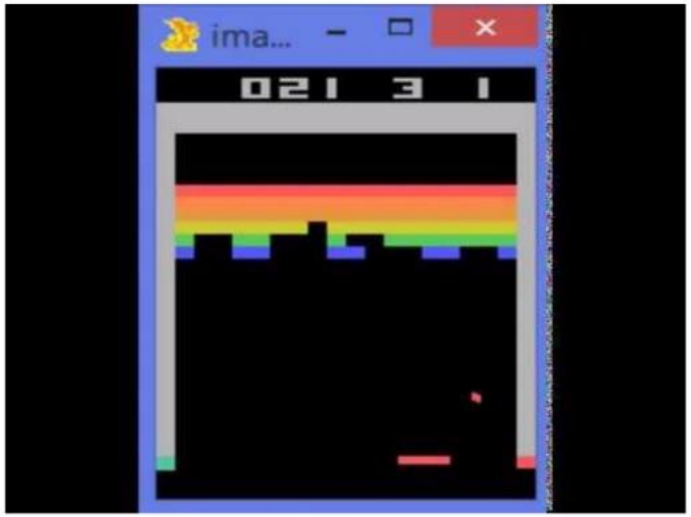


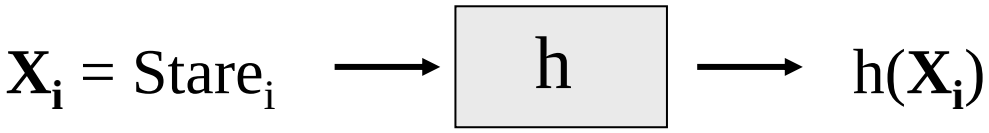
Figure: DeepMind Nature, 2015

Figure: Chelsea Finn, Sergey Levine, Pieter Abbeel

Model general invatare (vezi curs 1)

$T = \{\mathbf{X_1}, \mathbf{X_2}, \ldots, \mathbf{X_m}\}$ – **multime de invatare**

$f(\mathbf{X}) = ?$



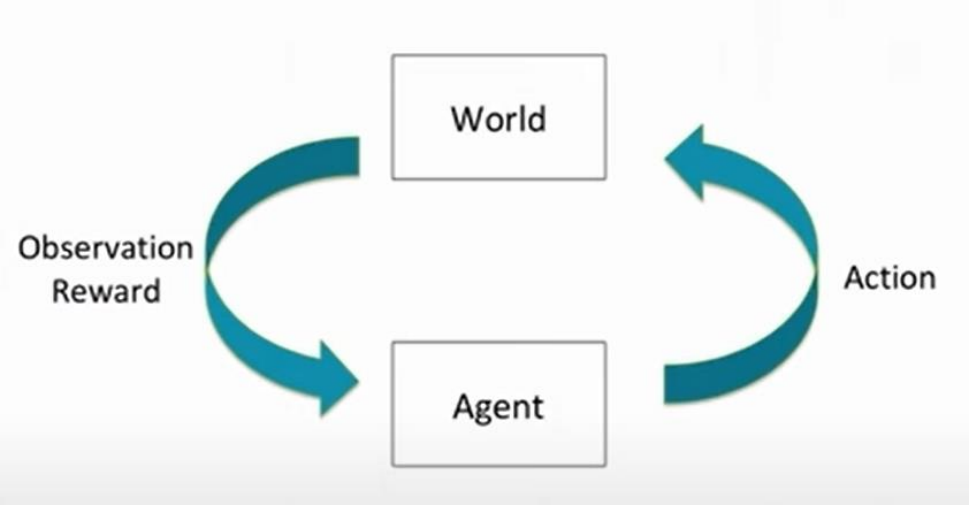
$h \in \mathbf{H}$

Invatare prin recompensa

- Se cunosc recompensele pentru $\mathbf{Stare_i}$, i=1,m
- $h(X_i)$ reprezinta actiunea de executat in starea X_i
- $f(\mathbf{X_i})$ – actiunea optima de executat in X_i
- Trebuie gasit **h** a.i. $h(\mathbf{X_i}) = f(\mathbf{X_i})$, $\forall \mathbf{Stare_i}$

RL - Introducere

- Reinforcement Learning (RL) = Invatarea prin recompensa
- Agentul invata o comportare in mediu pe baza interactiunilor cu mediul, prin explorari succesive
- Mediul ofera un feed back – **recompensa** = reinforcement



RL - Intro

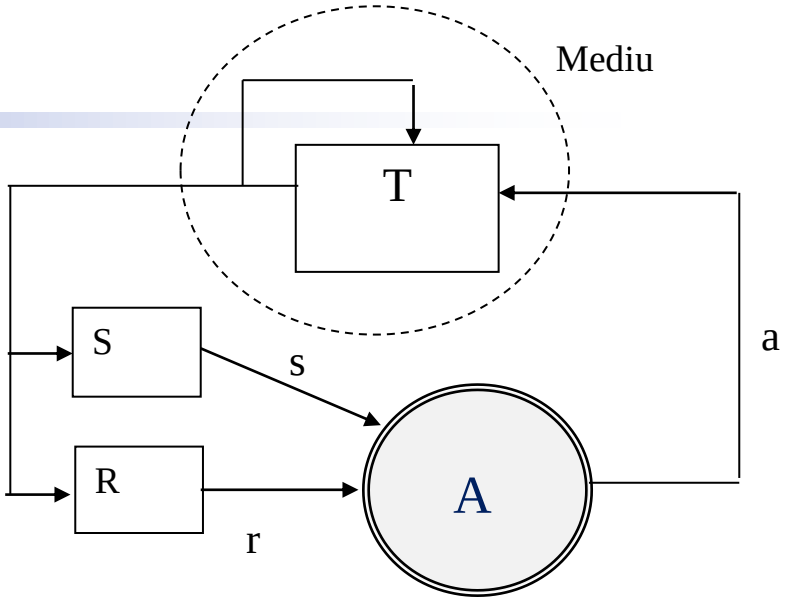
- **Invatare prin recompensa** = invata sa asocieze **actiuni** unor **stari** a.i. sa maximizam un ***semnal numeric de recompensa***.
- **Cum?** Incearca actiuni pt a descoperi acele actiuni care aduc o recompensa mai mare.
- Actiunile afecteaza **recompensa imediata** dar si urmatoarele stari, si prin aceastea, **recompensele viitoare** (**recompensa intarziata**)

RL - Caracteristici

- **Optimizare** – agentul invata sa ia decizii in conditii de incertitudine
- **Consecinte intarziate** – agentul nu stie imediat modul in care deciziile lui sunt asociate cu rezultate bune (temporal credit assignment problem)
- **Explorare** – agentul trebuie sa invete o serie de decizii si trebuie sa le invete prin explorare
- **Generalizare** – daca exista foarte multe stari
- **Agentul nu are un model al mediului**

Sistemul RL

A – agent RL
s – starea curenta a mediului
r – valoarea recompensei
(reinforcement signal)
a – actiunea agentului
T – modelul de evolutie a mediului



Modelul sistemului:

- o multime discreta de stari ale mediului S ($s \in S$)
- o multime discreta de actiuni ale agentului A ($a \in A$)
- o multime de semnale de recompensa, $r \in R$
- modelul de tranzitie al lumii, T (necunoscut de agent in RL)

- Pentru un mediu **nedeterminist, modelul tranzitiei mediului** este

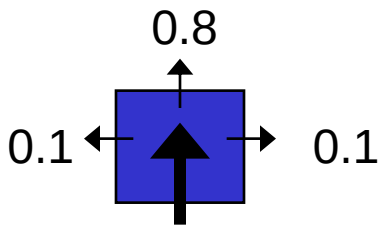
$$T : S \times A \rightarrow \boldsymbol{P}(S)$$

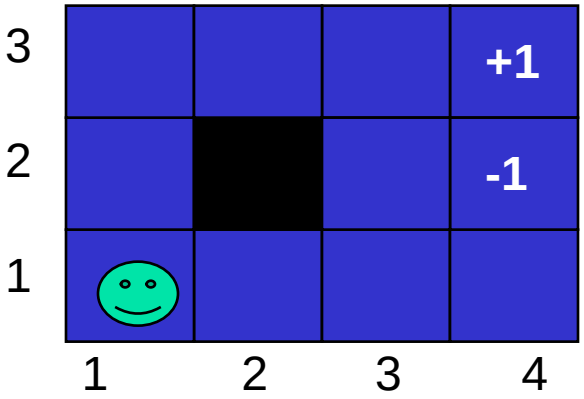
Notatii posibile **T(s, a, s') sau**

$$P^{a}_{s,s'} = \text{Prob}(\text{Urm}=\mathbf{s'} \mid \text{Curenta}=\mathbf{s}\text{ , actiune }\mathbf{a})$$

Exemplu mediu nedeterminist = grid 3 x 4

- Iesirea dorita se produce cu probabilitate de 0.8, si cu probabilitate de 0.2 (0.1, 0.1) agentul se muta in directii deviate cu 90^0 stanga sau dreapta.
- 2 stari terminale au recompensa **+1** si **–1**, toate celelalte stari au recompensa **–0.04**





Sus, Sus, Dreapta, Dreapta, Dreapta
(1,1) → (4,3)

Sistemul RL

- **Modelul sistemului**
  - Starile mediului S ($s \in S$)
  - Actiunile agentului A ($a \in A$)
  - Functia de recompensa, R: S x A $\rightarrow$ R
  - Modelul de tranzitie al lumii, T (necunoscuta de agent)
- **Politica agentului**
  - **π: S $\rightarrow$ A**
- Agentul trebuie sa gaseasca, prin experiente repetate, o **politica π** = **functie** care pune in corespondenta **stari** cu **actiuni**
- Politica trebuie sa **maximizeze o masura a recompensei pe termen lung – valoare sau utilitate**
- **Politica optima π^***

Sistemul RL

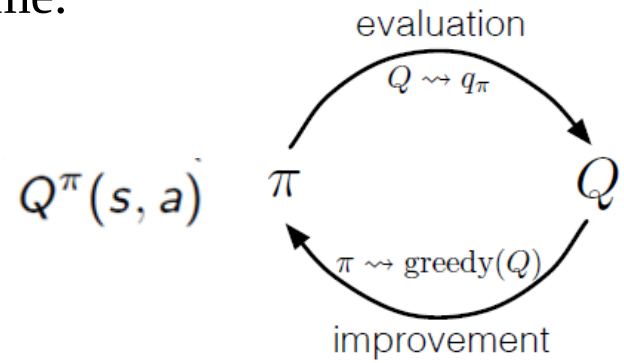
- **Valoarea** (functia valoare) sau **Utilitatea** unei stari **U(S)** – estimare pe termen lung a "cat de buna" este o stare
  - **Valoarea unei stari** este ***suma totala a recompensei*** pe care un agent spera sa o accumuleze in viitor, pornind de la o anumita stare.
  - Valorea unei stari poate fi vazuta ca **predictia recompensei pe termen lung**
  - Se cauta stari cu valori mari si nu cu recompense mari

Sistemul RL

- Mediu **este <u>nedeterminist</u>**
- Agentul nu are un **model al mediului** (T)
- Agentul invata numai **comportarea** sau **comportare si un model al mediului**
- Cum isi reprezinta agentul comportarea
  - **Utilitate/valoare a starilor** – **$U^{\pi}(s)$ / $V^{\pi}(s)$**
  - **Utilitate/valoare a perechilor stare-actiune – $Q^{\pi}(s,a)$**

RL - Probleme

- 2 probleme
- **Predictie** – evaluarea politicii – invatarea valorii unei stari sau a unei perechi stare-actiune pentru o politica data
- **Control** – gaseste / aproximeaza o politica optima
- Mentine o politica aproximativa si o functie valoare aproximativa. Functia valoare este modificata repetat pentru a se apropia de politica curenta si politica este imbunatatita repetat in raport cu functia valoare curenta. Impreuna se apropie de valorile optime.



2. Modele de comportare ale agentului RL

- **Modelul orizontului finit**: agentul parcurge un numar de *n* stari, ajunge in starea finala – ***episod***, apoi episodul se reia

$$U(S_t) = (\Sigma_{t=0,\ n}\ R(S_t,a))$$

$R(S_t)$ - recompensa primita in starea S_t cu actiunea a

U – recompensa estimata

- **Modelul orizontului infinit:** agentul traieste la nesfarsit si estimeaza recompensa pe termen lung

$$U(S_t) = (\Sigma_{t=0,\infty}\ R(S_t,a))$$

U – utilitatea / recompensa estimata pe termen lung

Modele de comportare ale agentului RL

- **Modelul orizontului infinit atenuat:** estimeaza recompensa pe termen lung dar recompensele viitoare sunt scazute cu un factor de atenuare δ.

$$U(S_t) = (\Sigma_{t=0,\infty} \delta^t R(S_t,a))$$

U – utilitate / recompensa estimata pe termen lung atenuata

$0 \leq \delta < 1$.

δ - dobanda, atenuarea, probabilitatea de a trai inca un moment de timp, artificiu matematic pt. a avea o limita la suma.

Exemplu - Orizont infinit atenuat

Recompensa atenuata

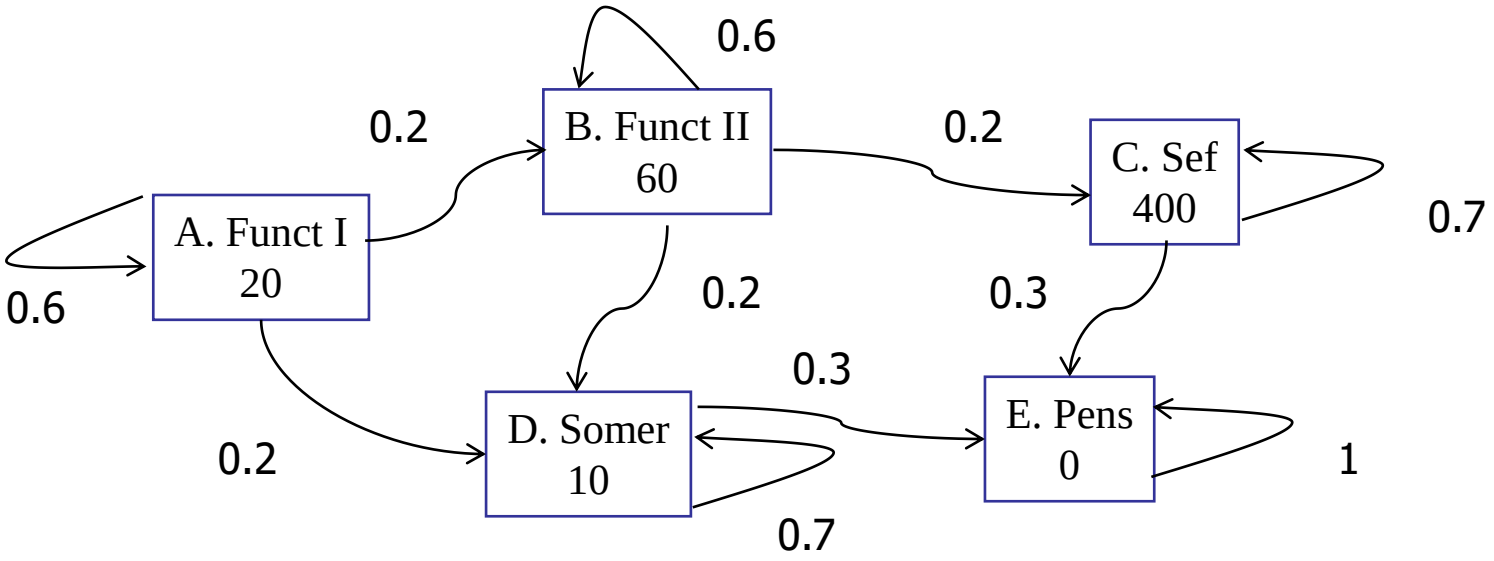
Un functionar este platit 20B pe an

Cat va castiga in total?

Cum calculam U(A) U(B) U(C) ....

Factor de atenuare δ

Suma atenuata a recompenselor viitoare (estimata)



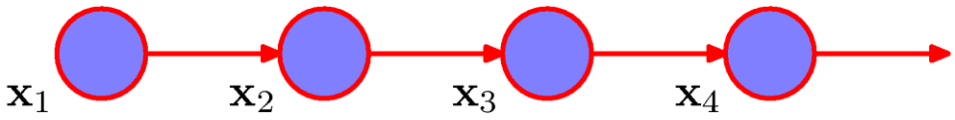
3. Sistem Markov

- Multime de stari **(S_1, S_2,…S_n)**
- Matrice de probabilitati de tranzitie (**$P : S \rightarrow \Pi(S)$**)

  P_{ij} = Prob(Urmatoare=S_j | Curenta = S_i)
- **Proprietatea Markov**

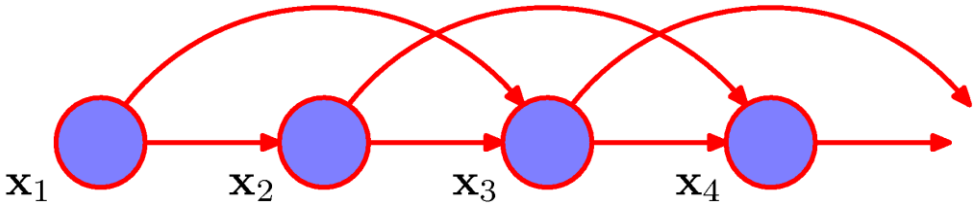
$$p(\mathbf{X}) = \prod_{i=1}^{n} p(X_n|X_{n-1})$$

1st order



$$p(\mathbf{X}) = \prod_{i=1}^{n} p(X_n|X_{n-1}, X_{n-2})$$

2nd order



Sistem Markov (MS) cu recompensa

- Multime de stari **(S_1, S_2,…S_n)**
- Matrice de probabilitati de tranzitie (**$P : S \rightarrow \Pi(S)$**)

  P_{ij} = Prob(Urmatoare=S_j | Curenta = S_i)
- Are proprietatea Markov de ordinul 1
- Fiecare stare are o recompensa r_1, r_2,…r_n (**$R: S \rightarrow R$**)
- Factor de atenuare $\delta \in [0,1]$
- In fiecare pas
  - Starea este S_i
  - Obtine recompensa r_i
  - Trece intr-o noua stare S_j cu probabilitatea P_{ij}
  - Toate recompensele viitoare sunt atenuate cu δ

Rezolvare sistem Markov cu recompensa - Determinarea utilitate stari

$U^*(S_i)$ = suma estimata atenuata a recompenselor viitoare incepand din starea S_i (valoare stare sau utilitatea)

$$U^*(S_i) = r_i(S_i) + \delta[P_{i1}U^*(S_1) + P_{i2}U^*(S_2) + .. + P_{in}U^*(S_n)] \quad i=1,n$$

$$\mathbf{U} = \begin{pmatrix} U^*(S_1) \\ U^*(S_2) \\ \dots \\ U^*(S_n) \end{pmatrix} \qquad \mathbf{R} = \begin{pmatrix} r_1 \\ r_2 \\ \\ r_n \end{pmatrix} \qquad \mathbf{P} = \begin{pmatrix} P_{11}\ P_{12} \dots P_{1n} \\ P_{21}\ \dots \\ \\ P_{n1}\ P_{n2} \dots P_{nn} \end{pmatrix}$$

$$\mathbf{U} = \mathbf{R} + \delta\ \mathbf{P}^*\mathbf{U} \qquad (\ r_i = R(S_i)\ )$$

Rezolva un sistem de *n* ecuatii cu *n* necunoscute

Exemplu - Orizont infinit atenuat

Un functionar este platit 20B pe an

Cat va castiga in total?

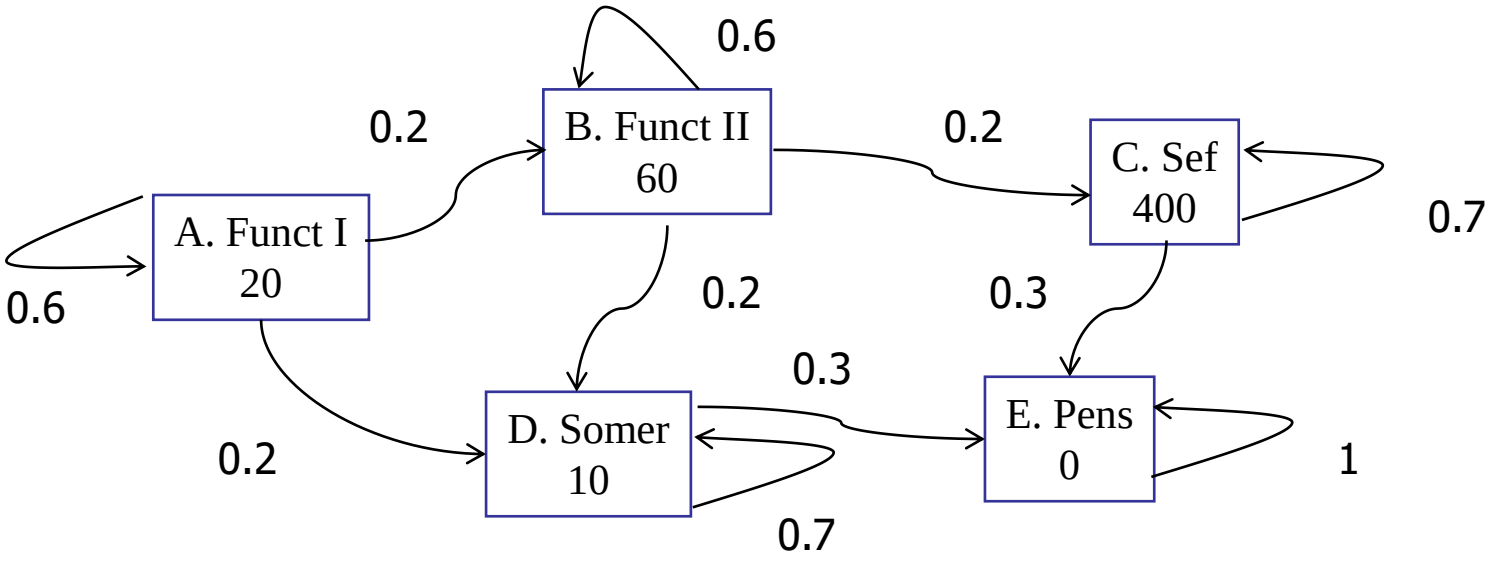
Factor de atenuare δ

Cum calculam U(A) U(B) U(C) ….

U(A) = 20 + δ[0.6*U(A) + 0.2*U(B) + 0.2*U(D)]

U(B) = 60 + δ[0.6*U(B) + 0.2*U(C) + 0.2*U(D)]

….



Iterarea valorii pentru MS cu recompensa

- **Rezolvarea sistemului de ecuatii**

Avantaj – obtine un raspuns exact

Dezavantaj – pt 100 000 stari se rezolva un sistem de 100 000 ecuatii cu 100 000 necunoscute

- **Rezolvarea iterativa - Value iteration (Iterarea valorii) pt MS**

$U^1(S_i)$, $U^2(S_i)$, $U^3(S_i)$ = …. $k \to \infty$ $U^k(S_i) \to U^*(S_i)$

1. Calculeaza $\mathbf{U^1(S_i)=r_i}$ pentru fiecare stare S_i, k←1

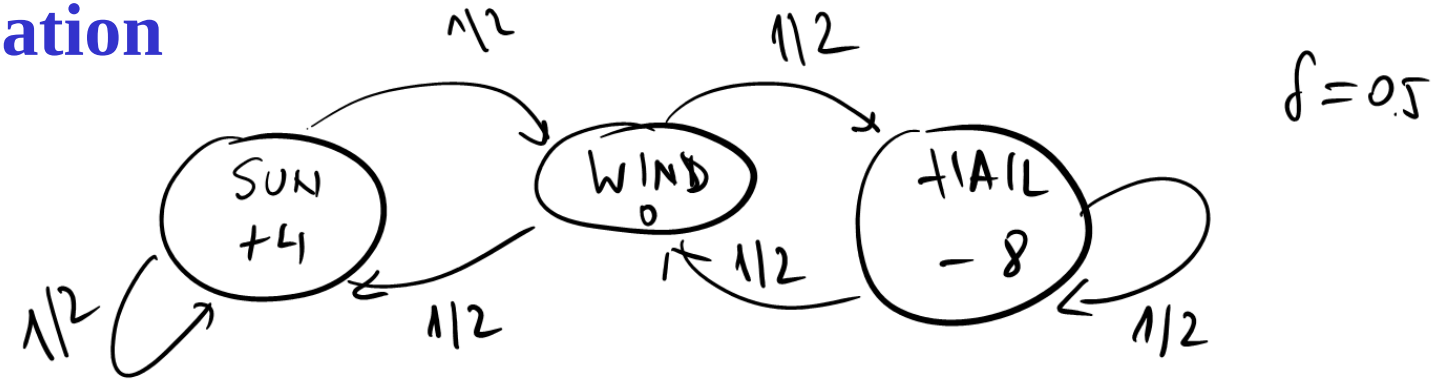
2. repeta

k ← k + 1

Calculeaza $\mathbf{U^k(S_i) = r_i + \delta \Sigma_{j=1,n} (P_{ij}U^{k-1}(S_j))}$ pentru fiecare stare S_i

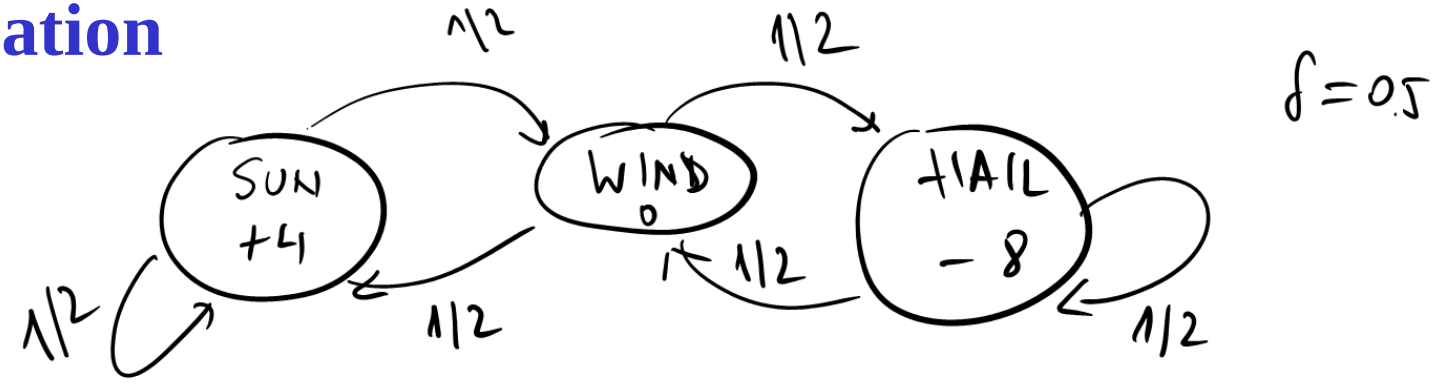
pana $\max_i |U^k(S_i) - U^{k-1}(S_i)| < eps$

Value iteration



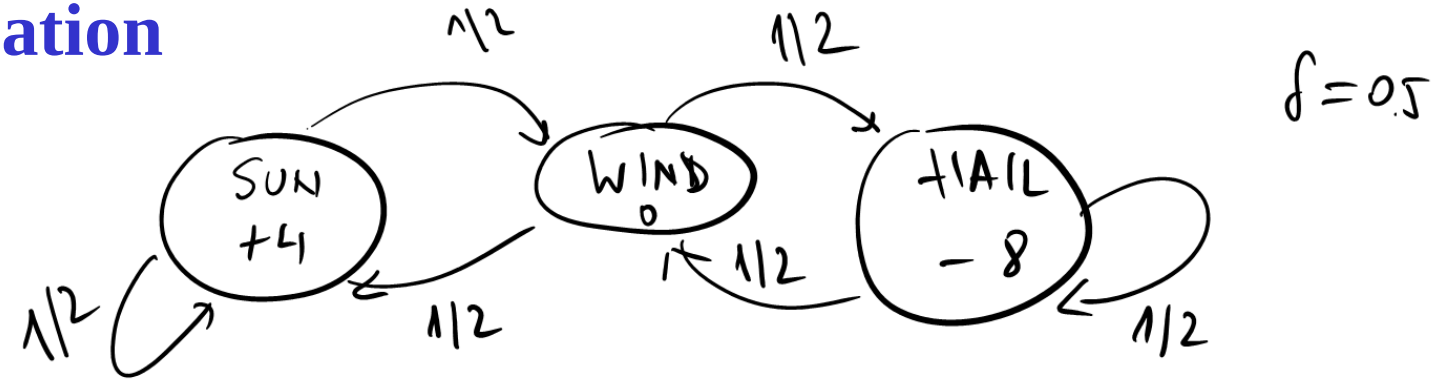
k	U^k(SUN)	U^k(WIND)	U^k(HAIL)
1	4	0	-8
2			
3			
4			
5			

Value iteration



k	U^k(SUN)	U^k(WIND)	U^k(HAIL)
1	4	0	-8
2	5	-1	-10
3			
4			
5			

Value iteration



k	U^k(SUN)	U^k(WIND)	U^k(HAIL)
1	4	0	-8
2	5	-1	-10
3	5	-1.25	-10.75
4	4.94	-1.44	-11
5	4.88	-1.52	-11.11

4. Proces de decizie Markov (MDP)

<S, A, P, R>

S – multime de stari

A – multime de actiuni

R – functia recompensa, **R: S x A → R**

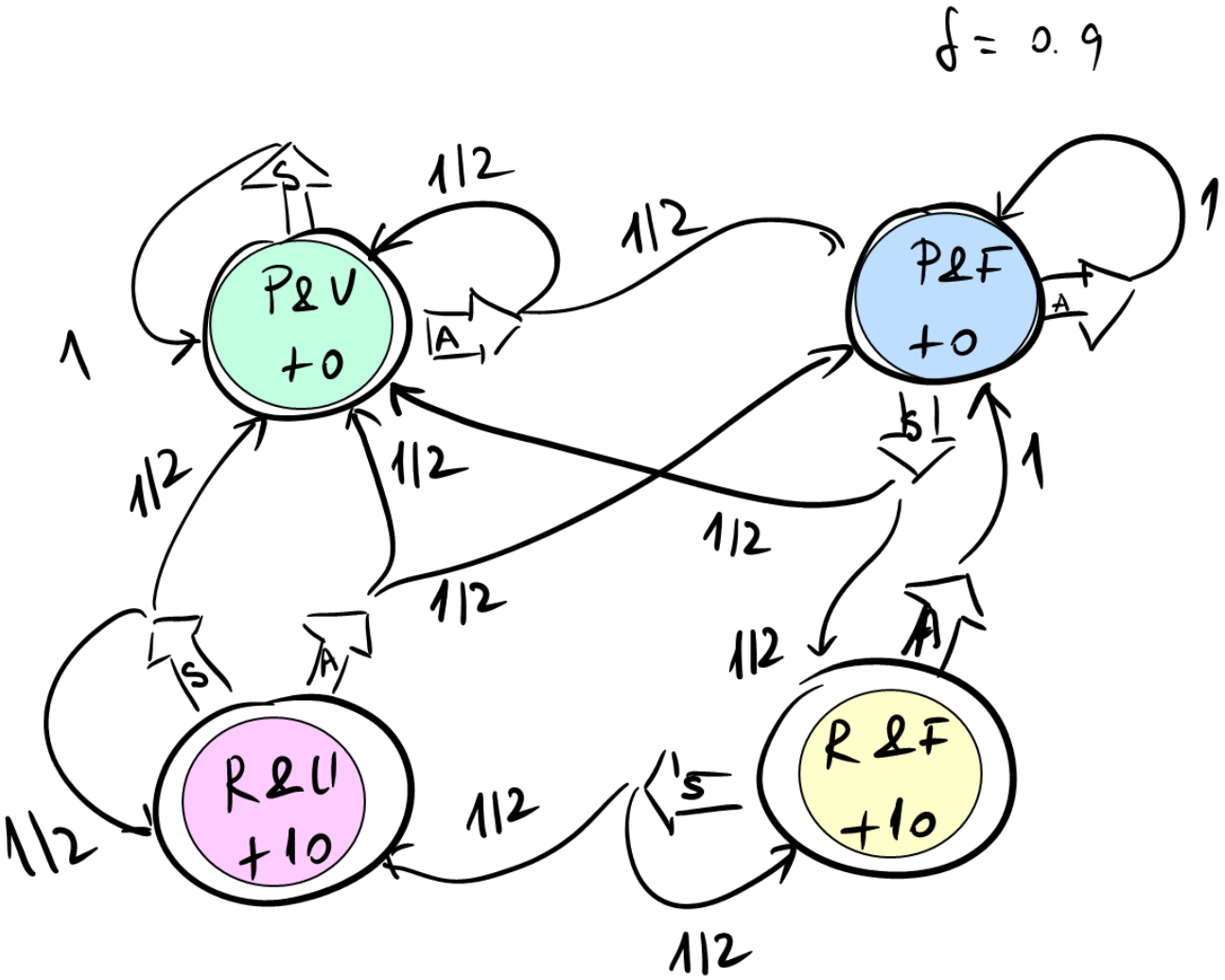
P : S x A → Π(S), functia de tranzitie probabilistica, cu Π(S) distributia de probabilitate peste S

$P^a_{s,s'}$=Prob(Urm=**s'**|Curenta=**s** si actiune **a**)

- In fiecare pas
  - Starea este **S**
  - Alege o actiune **a** (din $a_1 \ldots a_k$)
  - Trece in **S'** cu probabilitatea $\mathbf{P^a_{s,s'}}$
  - Obtine recompensa $\mathbf{R_i}$
  - Toate recompensele viitoare sunt atenuate cu **δ**

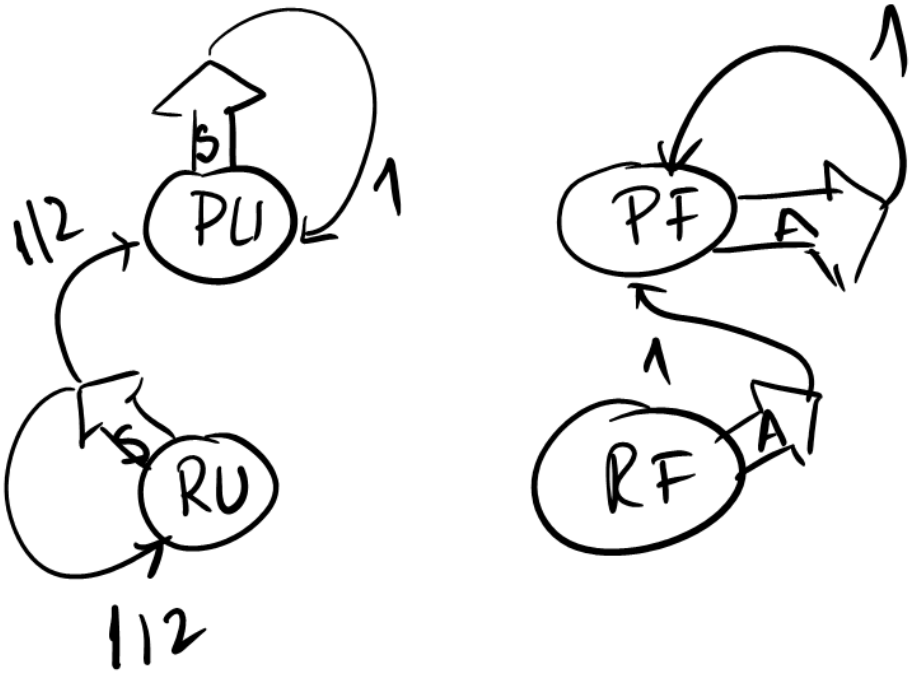
Vom folosi T(s,a,s') pentru $P^a_{s,s'}$

MDP



S – Save
A - Advertise

Politici



Politica 1

Stare	Actiune
PU	S
PF	A
RU	S
RF	A

Politica 2

Stare	Actiune
PU	A
PF	A
RU	A
RF	A

Markov Decision Process (MDP) – proprietati

- MDP: stari finite si actiuni finite / stari si actiuni infinte
- Alegerea unei politici π; alegerea unei politici optime π^*
- **Pentru fiecare MDP exista o politica optima**
- **Politica optima** = o politica a.i. din fiecare stare posibila nu exista o optiune mai buna decat sa urmeze acea politica

- Fiind dat un model **T, gasirea unei politici optime** = calculeaza **utilitatile** fiecarei stari $U(S_i)$ si foloseste utilitatea starilor pentru a selecta actiunea optima in fiecare stare **sau** gaseste **direct politica optima**

Ecuatiile Bellman - $U^{\pi}(s)$

- **Utilitatea unei stari** este recompensa imediata plus suma atenuata a recompenselor viitoare (estimate) din starea urmatoare, presupunand ca agentul selecteaza cea mai buna actiune (utilizeaza π^*)

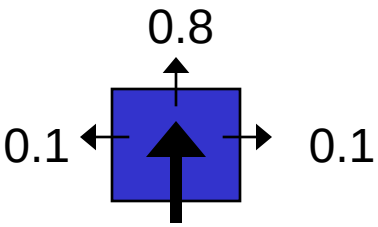
$$\mathbf{U(s) = \max_a [R(s,a) + \delta \Sigma_{s'} T(s,a,s') U(s')]}$$

- *Principle of Optimality: An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision. (Bellman, 1957)*

- Utilitatile starilor sunt solutiile acestor ecuatii

Mediu - grid 4 x 3

- Iesirea dorita se produce cu probabilitate de 0.8, si cu probabilitate de 0.2 (0.1, 0.1) agentul se muta in directii deviate cu 90^0 stanga sau dreapta.
- 2 stari terminale au recompensa +1 si –1, toate celelalte stari au recompensa –0.04
- $\delta = 1$



Recompense

3	-0.04	-0.04	-0.04	+1
2	-0.04		-0.04	-1
1		-0.04	-0.04	-0.04
	1	2	3	4

Utilitati/Valori

3	0.812	0.868	0.918	+1
2	0.762		0.660	-1
1	0.705	0.655	0.611	0.388
	1	2	3	4

Ecuatia Bellman pentru grid-ul 4x3

Ecuatia pentru starea (1,1)

$$U(1,1) = -0.04 + \delta \max\{\ 0.8\,U(1,2) + 0.1\,U(2,1) + 0.1\,U(1,1),$$

Up

$$0.9U(1,1) + 0.1U(1,2),$$

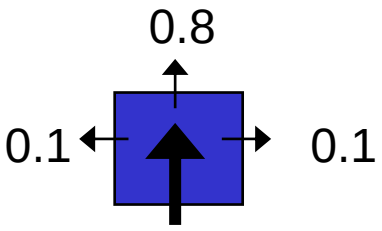
Left

$$0.9U(1,1) + 0.1U(2,1),$$

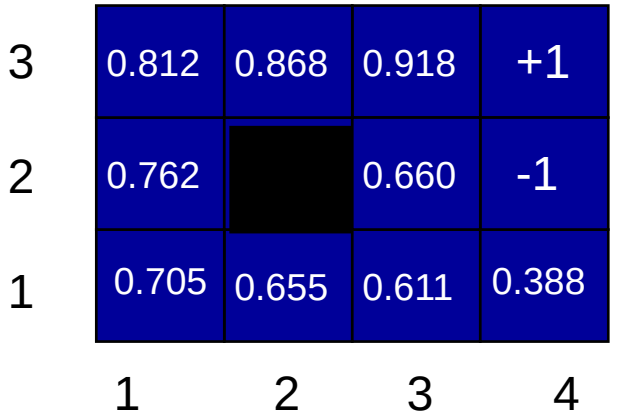
Down

$$0.8U(2,1) + 0.1U(1,2) + 0.1U(1,1)\}$$

Right



Up este cea mai buna actiune



2 stari terminale au recompensa +1 si –1
toate celelalte stari au recompensa –0.04
$\delta = 1$

Value iteration pt MDP

$U^1(S_i)=R(S_i)$

$U^2(S_i) = \max_a [\, R(S_i,a) + \delta \Sigma_{j=1,n}\, T(S_i,a,S_j)\, U^1(S_j) \,], \quad S_j=succ(S_i)$

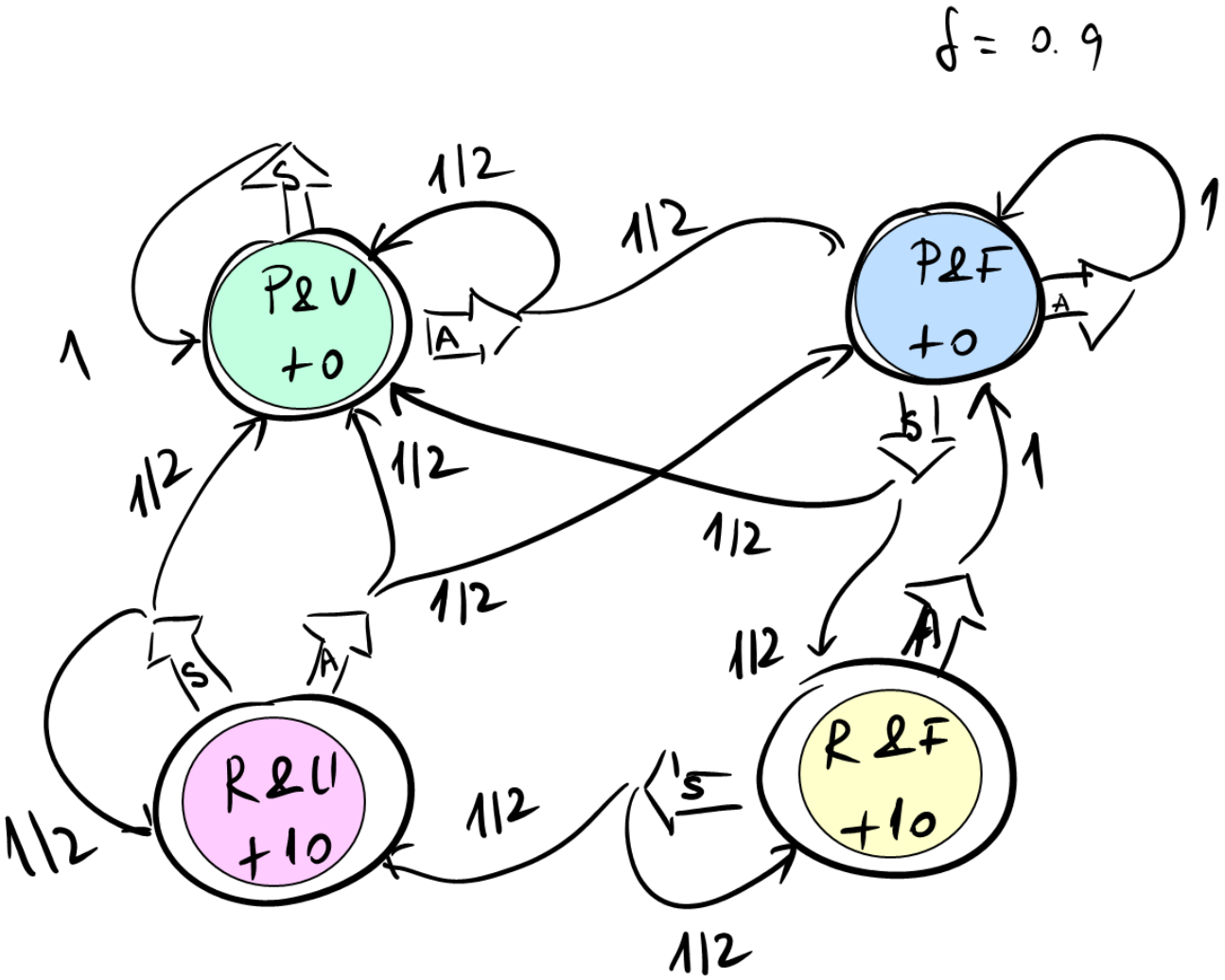
….

$U^{k+1}(S_i) = \max_a [\, R(S_i,a) + \delta \Sigma_{j=1,n}\, T(S_i,a,S_j)\, U^k(S_j)]$

$\mathbf{k \to \infty \;\; U^k(S_i) \to U^*(S_i)}$

1. Calculeaza $U^1(S_i)$ pentru fiecare stare S_i , $k \leftarrow 1$
2. **repeta**
   $k \leftarrow k+1$
   Calculeaza $U^k(S_i)$ pentru fiecare stare S_i
   pana $\max_i |U^k(S_i) - U^{k-1}(S_i)| < eps$

MDP



S – Save
A - Advertise

Value iteration pt MDP

k	U^k(PU)	U^k(PF)	U^k(RU)	U^k(RF)
1	0	0	10	10
2	0	4.5	14.5	19
3	2.03	6.53	25.08	25.07
4	3.852	12.20	29.63	??
5	7.22	15.07	32.00	??
6	10.03	17.65	33.58	??

?? – sa se calculeze

Gasirea politicii optime

- Daca cunoastem $U(S_i)$ pentru toate starile S_i
- Defineste "cea mai buna actiune" in S_i ca fiind

$$\mathbf{arg\ max_a\ [R(S_i,a) + \delta\ \Sigma_{j=1,n} T(S_i,a,S_j)\ U(S_j)]}$$

Maximum Expected Utility (MEU)

politica optima

$$\mathbf{\pi^*(s) = arg\ max_a\ [R(s,a) + \delta\ \Sigma_{s'} T(s,a,s')\ U^*(s')]}$$

Policy iteration pt MPD

Fie $\pi_t(S)$ actiunea selectata in starea **S** la iteratia **t**

1. t=0
2. Alege o politica oarecare π_0 (aleator)
3. Pentru fiecare S_i, calculeaza $U^0(S_i)$ recompensa pe termen lung obtinuta pornind din starea S_i, folosind π_0

$$U^0(S_i) = R(S_i,a) + \delta \Sigma_{j=1,n} T(S_i, \pi_0(S_i),S_j) U^0(S_j)$$

(Pe baza ei se va calcula

$$\pi_t(S_i) = \arg\max_a [R(S_i,a) + \delta \Sigma_{j=1,n} T(S_i, a, S_j) U^0(S_j)])$$

Policy iteration pt MPD

4. repeta

- $t \leftarrow t + 1$
- Calculeaza politica $\pi_t(s)$

$$\pi_t(S_i) = \arg\max_a [R(S_i,a) + \delta \Sigma_{j=1,n} T(S_i, a, S_j) U^{t-1}(S_j)]$$

- Calculeaza recompensa pe termen lung obtinuta pornind din starea s, folosind π_t

$$U^t(S_i) = R(S_i,a) + \delta \Sigma_{j=1,n} (T(S_i, \pi_t(S_i),S_j) * U^{t-1}(S_j))$$

pana $\pi_t = \pi_{t-1}$

5. Reinforcement learning

- **Daca agentul are un model complet al mediului** (stie MDP) – determina utilitatea starilor pentru a afla politica optima sau direct politica
- **Daca agentul NU are un model complet al mediului -** exploreaza mediul si utilizeaza recompensa pentru **a invata utilitati ale starilor (sau utilitati ale perechilor stare-actiune) sau politici** pentru acest mediu – **Reinfocement Learning**

RL – Metode

- **Invatarea bazata pe model (model-based learning)** – incearca sa invete un model explicit al mediului si foloseste modelul invatat pentru a gasi utilitatea starilor sau politica optima
- **Invatare fara model (model-free learning)** – invata direct utilitatea starilor sau a perechilor stare-actiune si obtine politica optima
- **Predictie / Evaluare**
  - Agentul urmeaza o politica si invata utilitatea starilor sau a perechilor stare-actiune
- **Control**
  - Agentul trebuie sa invete si ce actiuni sa faca

Model-based learning

- **Valoarea unei stari urmand o politica π este utiltatea acelei stari** (long term discounted reward)
- Folosim ec. Bellman
- **$U(s) = \max_a [R(s,a) + \delta \Sigma_{s'} T(s,a,s') U(s')]$**
- DAR agentul NU cunoaste $T(S_i,a,S_j)$
- Incearca sa estimeze T

Pentru a invata, exploreaza mediul

- Ce se stie prin explorare?
- O secventa de stari si recompensa asociata, de ex.

$S_1(R=0) \rightarrow S_2(R=0) \rightarrow S_3(R=4) \rightarrow$

$S_4\ (R=0) \rightarrow S_5\ (R=0)$

- Pe baza acestei secvente sa se estimeze $U^*(S_1)$, $U^*(S_2)$ …
- La MDP stiam probabilitatile de tranzitie intre stari pt a calcula U*
- Aici nu le stim; trebuie sa calculam U* numai pe baza observatiilor pe unde trece si ce recompensa primeste

5.1 Invatare ADP

- **ADP – Adaptive Dynamic programming**
- Model-based
- Metoda Monte Carlo
- Evalueaza utilitatile prin explorare
- Politica agentului este fixa: in starea **s** se executa actiunea $\pi(s)$: $S \rightarrow A$
- Agentul invata cat de buna este o politica, adica $U^{\pi}(S)$
- Aplicata la taskuri episodice (modelul orizontului finit)

- Estimeaza probabilitatile tranzitiilor T(s,a,s') pe baza frecventei cu care se ajunge in **s'** dupa ce se executa **s** in **a**

  exp1: $s_1 \ldots s_3 \rightarrow^a s_4 \ldots$

  exp2: $s_1 \ldots s_3 \rightarrow^a s_4 \ldots$

  exp3: $s_1 \ldots s_3 \rightarrow^a s_5 \ldots$

  $$T(s3, a, s4) = 2/3$$

- Utilizeaza Bellman pentru politica fixa

  $$\mathbf{U^{\pi}(s) = R(s,a) + \delta\,(\Sigma_{s'} T(s, \pi(s), s')\, U^{\pi}(s'))}$$

- Se utilizeaza pentru un comportament episodic al agentului (modelul orizontului finit)

ADP (Adaptive Dynamic Programming)

function ADP-Agent(perceptie) **returns** o actiune

inputs: perceptie care indica starea curenta **s'** si recompensa **r'**

variable: π, a politica fixa

mdp, MDP cu model T, recompense R, atenuare δ

U, o tabela de utilitati, initial vida

N_{sa}, o tabela de frecvente pentru perechi **stare-actiune**, initial zero

$N_{s'|as}$ o tabela de frecvente a **iesilor** pentru perechi **stare-actiune**, initial zero

s, a, starea si actiunea anterioara, initial nule

if s' este stare noua **then** U[s'] $\leftarrow$ r', R[s'] $\leftarrow$ r'

if s nu este nula **then**

incrementeaza N_{sa}[s,a] si $N_{s'|sa}$[s',s, a]

for fiecare stare **t** a.i. $N_{s'|sa}$[t, s, a] <>0 **do**

T[s,a,t] $\leftarrow$ $N_{s'|sa}$[t,s,a] / N_{sa}[s,a]

U $\leftarrow$ Evaluare(π,U)

if s' este stare finala **then** **s, a** $\leftarrow$ null **else** **s, a** $\leftarrow$ s', π[s']

return a

end

Evaluare
De ex foloseste Iterarea valorii
$U^{k+1}(S_i) = \max_a(R(S_i,a) + \delta \Sigma_{j=1,n} T(S_i,a,S_j) U^k(S_j))$

5.2 Invatare TD (Temporal difference)

- **Model-free**
- Agent pasiv - politica agentului este fixa: in starea **s** se executa actiunea $\pi(s)$: $S \rightarrow A$
- Se folosesc tranzitiie observate pentru a corecta valorile starilor prin care se trece
- Foloseste estimarea valorii starii urmatoare pentru a actualiza estimarea valorii starii curente (procedura de **bootsrapping**)
- Se memoreaza tabela U^{est}
- Poate fi utilizat atat pentru taskuri episodice cat si pentru taskuri continue (modelul orizontului infinit)

Invatare TD (Temporal difference)

exp1: $s_0 \rightarrow \ldots. s_i \rightarrow s_j \ldots$ si observam $s \rightarrow s'$

- Actualizam $\mathbf{U^{\pi}(s)}$ in functie de $\mathbf{R(s)}$ si $\mathbf{\delta\, U^{\pi}(s')}$
- Valoarea estimata a starii curente este actualizata a.i sa fie mai aproape de valoarea estimata a starilor viitoare
- Estimeaza valoarea unei stari pentru o politica data dar nu imbunatateste politica
- Metoda TD cea mai simpla TD(0) / one step TD

- Se introduce un parametru - α ***rata de invatare***

$$\mathbf{U^{\pi}(s) = U^{\pi}(s) + \alpha_t(R(s) + \delta\, U^{\pi}(s') - U^{\pi}(s))}$$

- α estimeaza eroarea intre valoarea starii curente si valoarea reala a acelei stari, pentru o politica data (un fel de aproximare stohastica)

Temporal difference learning

$$U^{\pi}(s) = U^{\pi}(s) + \alpha_t(R(s) + \delta\, U^{\pi}(s') - U^{\pi}(s))$$

TD target

- TD error = $R(s) + \delta\, U^{\pi}(s') - U^{\pi}(s))$
- α sa descreasca lent in timp (α_t - scade pe masura ce numarul de vizitari ale unei stari creste)
- $\Sigma_{t=1,\infty}\ \alpha_t = \infty$
- $\Sigma_{t=1,\infty}\ (\alpha_t)^2 < \infty$

Temporal difference learning

function TD-Agent(perceptie) **returns** o actiune
inputs: perceptie care indica starea curenta **s'** si recompensa **r'**
variable: **π,** o politica fixa
U, o tabela de utilitati, initial goala
Ns, o tabela de frecvente pentru **stari**, initial zero
s, a, r starea, actiunea si recompensa anterioara, initial nule

if **s'** este stare noua **then** U[s'] ← r'

if **s** nu este nula **then**

incrementeaza Ns[s]

U[s] ← U[s] + α(Ns[s])*(r + δ U [s'] – U [s])

if **s'** este stare finala

then **s, a, r** ← null

else **s, a, r** ← s', π[s'], r'

return a

end

s, a, r ← s', π[s'], r'

Taskuri episodice sau continue

Temporal difference learning

- Actualizeaza estimarea pe baza altor estimari
- Bootstrapping
- Pt o politica fixa se demonstreaza ca va converge la $\mathbf{U^{\pi}(s)}$
- Pentru ca metoda sa fie convergenta, adica ca $U^{\pi}(S) \rightarrow U^{*}(S)$ pentru orice stare, este necesar ca toate starile sa fie vizitate la fel de des
- Nu necesita un model
- Mediul ofera conexiunea intre stari adiacente sub forma tranzitiilor observate
- TD poate fi implementata on-line

ADP vs TD

- ADP si TD – ambele incearca sa modifice estimarea utilitatii a.i. fiecare stare sa fie "in concordanta" cu starea ei succesoare
- TD mentine concordanta starii curente numai cu starea urmatoare
- ADP mentine concordanta cu toti succesorii posibili ai starii curente, ponderat cu probabilitati
- TD face o modificare pe ciclu
- ADP face mai multe modificari pe ciclu a.i. sa mentina consistenta utilitatilor estimate cu mediul
- TD – taskuri continue sau episodice
- ADP – taskuri episodice (are stari finale, explorarea trebuie reluata cu pastrarea valorilor)

5.3 Q-learning

- **Model-free**, Agent activ
- Invata utilitati ale perechilor actiune-valoare in loc de utilitati ale starilor
- **Functie actiune-valoare** = atribuie o utilitate estimata executarii unei actiuni intr-o stare
- Q: A x S $\rightarrow$ U
- **Q(a, s) –** suma estimata atenuata a recompenselor viitoare obtinute incepand din starea **s**, alegerea actiunii **a** si urmarind apoi o *politica optima*
- Valorile Q sunt legate de utilitatea starilor astfel

$$\mathbf{U(s) = \max_a Q(a, s)}$$

Q-learning

- $U(s) = \max_a Q(a, s)$
- Bellman

  $U(s) = \max_a [R(s,a) + \delta \Sigma_{s'} T(s,a,s') U(s')]$
- Daca am extrapola ec Bellman, am obtine

  $Q(a,s) = R(s,a) + \delta \Sigma_{s'} (T(s, a, s') \max_{a'} Q(a', s'))$

  Dar necesita un model, deci nu se poate folosi
- Agentul invata dar NU are un model al mediului
- **Solutie: Se utilizeaza aceeasi abordare ca la TD**
- In loc de U, mentine o tabela cu valorile Q

Q-learning

In TD am utilizat

$U^{\pi}(s) = U^{\pi}(s) + \alpha_t(R(s,a) + \delta\, U^{\pi}(s') - U^{\pi}(s))$

In Q-learning se utilzeaza

$Q(a,s) = Q(a,s) + \alpha_t(R(s,a) + \delta \max_{a'} Q(a', s') - Q(a,s))$

calculata dupa fiecare tranzitie din s in s'.

α - viteza de invatare

Q-learning

function Q-Learning-Agent(perceptie) **returns** o actiune

inputs: percept, o perceptie ce indica starea curenta s′ si recompensa r′

variable: **Q,** o matrice cu valori pentru perechi stare actiune, initi. arbitrar

Nsa, o tabela de frecvente pentru perechi stare-actiune

s, a, r starea, actiunea si recompensa anterioare

if **s** <> null **then**

incrementeaza Nsa[s,a]

$Q[a,s] \leftarrow Q[a,s] + \alpha(Nsa[s,a]) * (r + \delta \max_{a'} Q[a',s'] - Q[a,s])$

s, a, r $\leftarrow$ s′, $\text{argmax}_{a'} Q[a', s']$, r′

if s′ este stare finala **then** **s, a, r** $\leftarrow$ null

else **s, a, r** $\leftarrow$ s′, $\text{argmax}_{a'} Q[a', s']$, r′

return a

end

s, a, r $\leftarrow$ s′, **f**($\text{argmax}_{a'} Q[a', s']$, a′), r′

explorare - exploatare

$\forall s \in S \ \pi(s) = \text{argmax}_a Q(a,s)$

Explorare vs exploatare

$$Q(a,s) = Q(a,s) + \alpha_t(R(s,a) + \delta \max_{a'} Q(a', s') - Q(a,s))$$

ε-greedy policy: $\text{argmax}_{a'} Q[a',s']$ cu prob (1-ε) altfel un a' aleatory, cu $0 < \varepsilon < 1$

$s, a, r \leftarrow s', \mathbf{f}(\text{argmax}_{a'} Q[a', s'], a'), r'$

Varianta 1 - ε descreste liniar in timp

Varianta 2

Distributie Boltzman prin care se controleaza explorare vs. exploatare printr-un parametru de temperatura τ (softmax policy)

$$\varepsilon = \frac{e^{Q(s,a)/\tau}}{\sum_{a' \in A} e^{Q(s,a')/\tau}}$$

Viteza de invatare

$$Q(a,s) = Q(a,s) + \alpha_t(R(s,a) + \delta \max_{a'} Q(a', s') - Q(a,s))$$

- Aceleasi conditii de convergenta
  - Toate perechile stare-actiune sunt vizitate la fel de des
  - α_t descreste lent in timp
  - $\Sigma_{t=1,\infty} \alpha_t = \infty$
  - $\Sigma_{t=1,\infty} (\alpha_t)^2 < \infty$

Comentariu despre politici

- **On-policy learning**
- **Off-policy learning**

5.4 SARSA

In TD am utilizat

$U^{\pi}(s) = U^{\pi}(s) + \alpha_t(R(s,a) + \delta\, U^{\pi}(s') - U^{\pi}(s))$

- **SARSA** (s, a, r, s',a')

$Q(a,s) \leftarrow Q(a,s) + \alpha(R(s,a) + \delta\, Q(a', s') - Q(a,s))$

- Extindere directa a TD Learning
- Este on-policy learning

Comparatie Q-learning cu SARSA

- **Q-learning – off-policy learning**

  Estimeaza direct valoarea politicii in timp ce actioneaza conform unei alte politici
- Mentine estimari Q ale perechilor stare-actiune si le foloseste pentru a gasi valoarea celei mai bune actiuni urmatoare

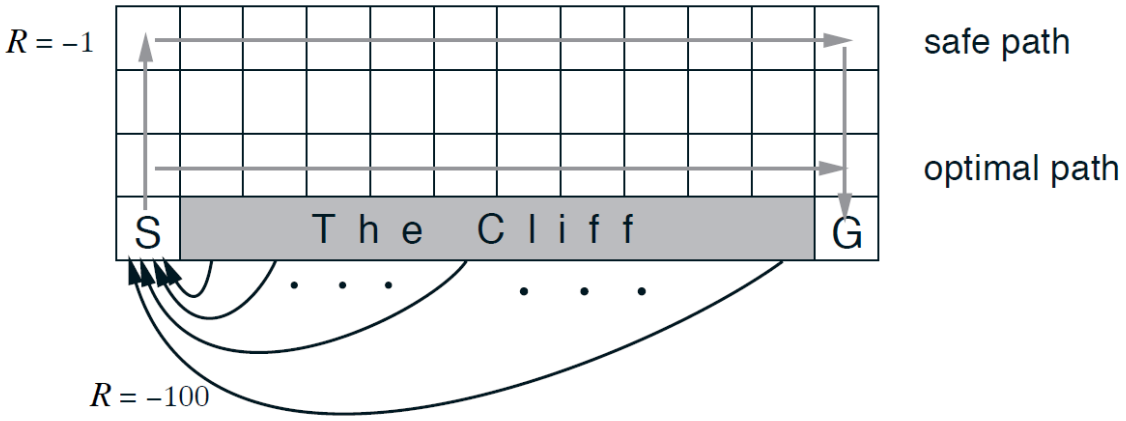
$$Q(a,s) \leftarrow Q(a,s) + \alpha(R(s,a) + \delta \max_{a'} Q(a', s') - Q(a,s))$$

- **SARSA – on-policy learning**
- estimeaza valoarea comportarii/politicii curente si apoi actualizeaza aceasta politica incercand sa estimeze

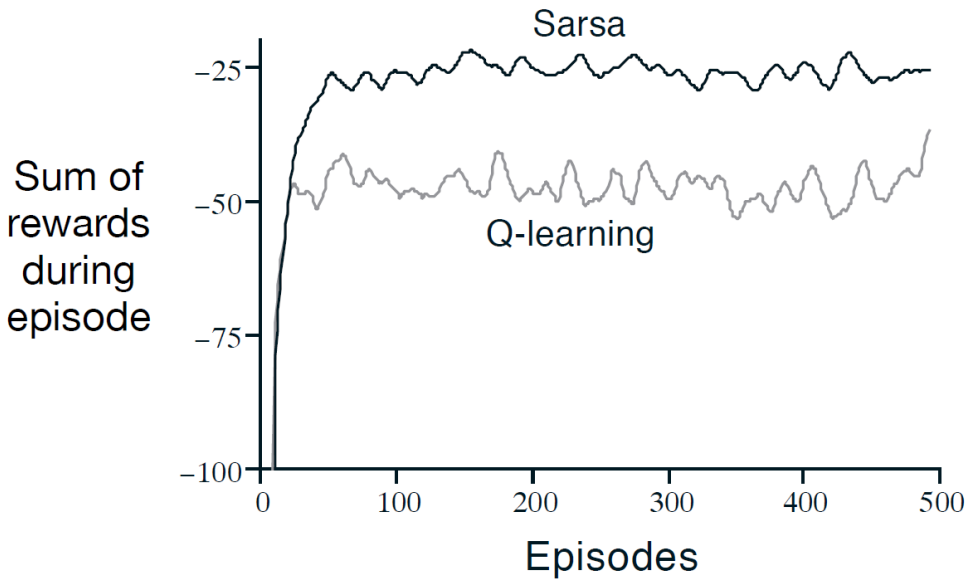
$$Q(a,s) \leftarrow Q(a,s) + \alpha(R(s,a) + \delta Q(a', s') - Q(a,s))$$

Cliff Walking – **comparatie SARSA vs Q-learning**

(exemplu Sutton)



Q-learning cu
ε-greedy policy
ε=0.1
fara discount



5.5 TD(n)

- Extinderea TD la mai multi pasi
- TD(0) – one-step – fac actualizari 1 pas inainte
- TD(n) – fac actualizari mai multi pasi inainte

$$U(s_t) \leftarrow U(s_t) + \alpha(R(s_t,a) + \delta U^{\pi}(s_{t+1}) + \delta^2 U^{\pi}(s_{t+2}) + \ldots + \delta^n U^{\pi}(s_{t+n}) - U^{\pi}(s))$$

- La fel pentru Q-learning
- $Q(a_t,s_t) \leftarrow Q(a_t,s_t) + \alpha(R(s_t,a) + \delta \max_{at+1} Q(a_{t+1},s_{t+1}) + \delta^2 \max_{at+2} Q(a_{t+2},s_{t+2}) + \ldots\ldots - Q(a,s))$

Double Q-Learning

- Utilizeaza 2 modele de Q-learning, Q1 si Q2, si foloseste unul ca target pentru celalalt

---

Initialize $Q_1(s, a)$ and $Q_2(s, a), \forall s \in S, a \in A$ $t = 0$, initial state $s_t = s_0$

loop

Select a_t using ϵ-greedy $\pi(s) = \arg\max_a Q_1(s_t, a) + Q_2(s_t, a)$

Observe (r_t, s_{t+1})

if (with 0.5 probability True) **then**

$$Q_1(s_t, a_t) \leftarrow Q_1(s_t, a_t) + \alpha(r_t + Q_1(s_{t+1}, \arg\max_{a'} Q_2(s_{t+1}, a')) - Q_1(s_t, a_t))$$

else

$$Q_2(s_t, a_t) \leftarrow Q_2(s_t, a_t) + \alpha(r_t + Q_2(s_{t+1}, \arg\max_{a'} Q_1(s_{t+1}, a')) - Q_2(s_t, a_t))$$

end if

$t = t + 1$

end loop

7. Generalizare RL

- Problema invatarii in spatii mari de stari – f multe stari
- Tehnici de generalizare – stocare compacta a ceea ce s-a invatat si gruparea starilor "similare"
- U(stare) = U(stari similare)

Problema unui numar mare de stari - RL

- Metode tabulare
- Generalizare prin Functii de aproximare
  - Combinatie liniara de caracteristici
  - Retele neurale
  - Arbori de decizie
  - Nearest neighbors
  - Transformate fourier

Invatare automata

Universitatea Politehnica Bucuresti

Anul universitar 2021-2022

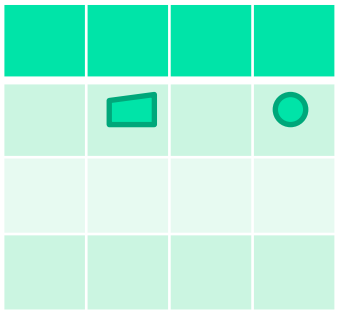
Adina Magda Florea

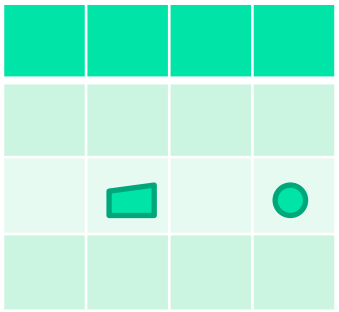
Curs nr. 8

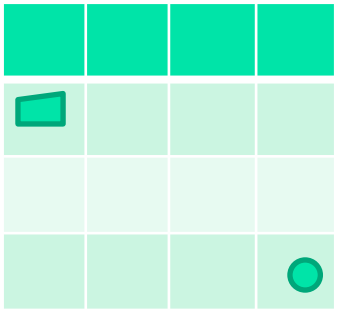
- Deep Q-Learning

Aproximarea functiei valoare

- Value Function Approximation (VFA)
- Reprezentarea functiei valoare (a unei stari – V(s)/U(s) sau a unei perechi stare-actiune Q(s,a)) printr-o functie parametrizata (in loc de o tabela)
- Generalizare la stari care nu au fost vizitate sau nu au fost vizitate suficient de des



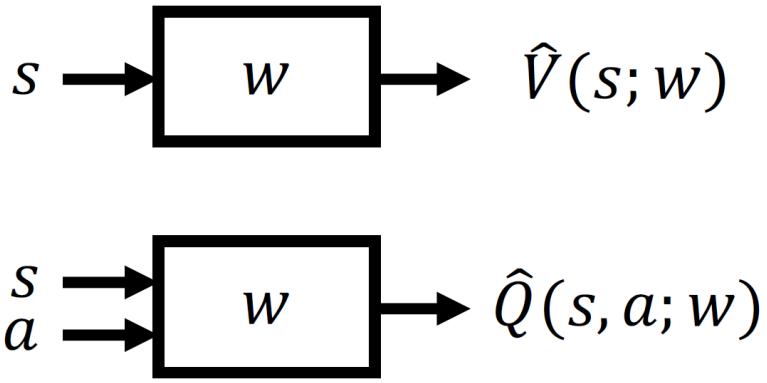




Aproximarea functiei valoare

- Reprezinta starile ca o multime de caracteristici **x(s)** impreuna cu un set de parametrii **w**
- Gaseste parametrii **w** prin minimizarea erorii (loss) intre valoarea *V(s)* sau *Q(s,a)* si valoarea functiei de aproximare $\hat{V}(s;w)$ sau $\hat{Q}(s,a;w)$

$$\begin{bmatrix} x_1(s) \\ .. \\ x_n(s) \end{bmatrix} \quad \begin{bmatrix} w_1 \\ .. \\ w_n \end{bmatrix}^T$$

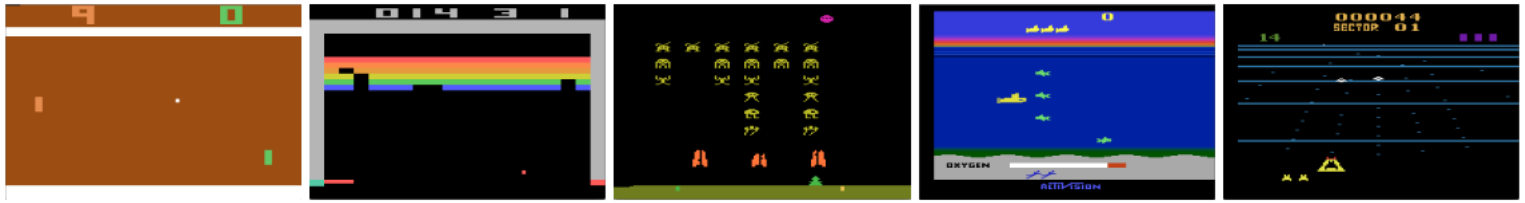


Aproximarea functiei valoare

- S-a incercat mult timp cu VFA liniare
- Probleme:
  - reprezentarea caracteristicilor
  - VFA + off-policy+ bootstrapping poate diverge
- NN multi-strat au fost utilizate in RL inca din anii 90, de ex in jocul de table

Utlizarea DNN in RL

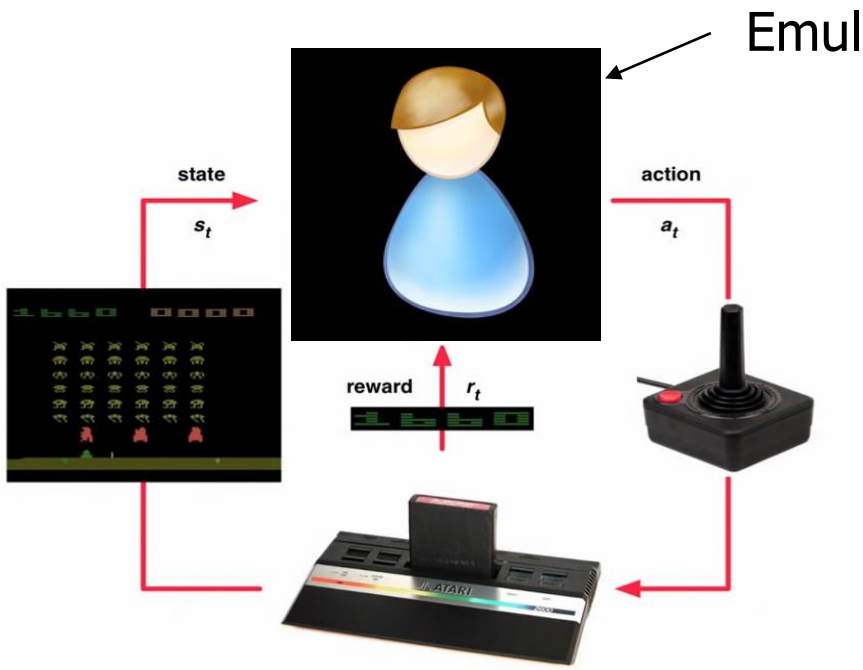
- Google Deep Mind – a aratat ca o DNN poate automatize procesul de reprezentare a caracteristicilor (2013, 2015)
- DQN – Q-learning with CNN – 49 de jocuri Atari prin interactiunea cu un emulator de jocuri



Pong, Breakout, Space Invaders, Sea Quest, Beam Rider

Deep Q-Networks (DQN)

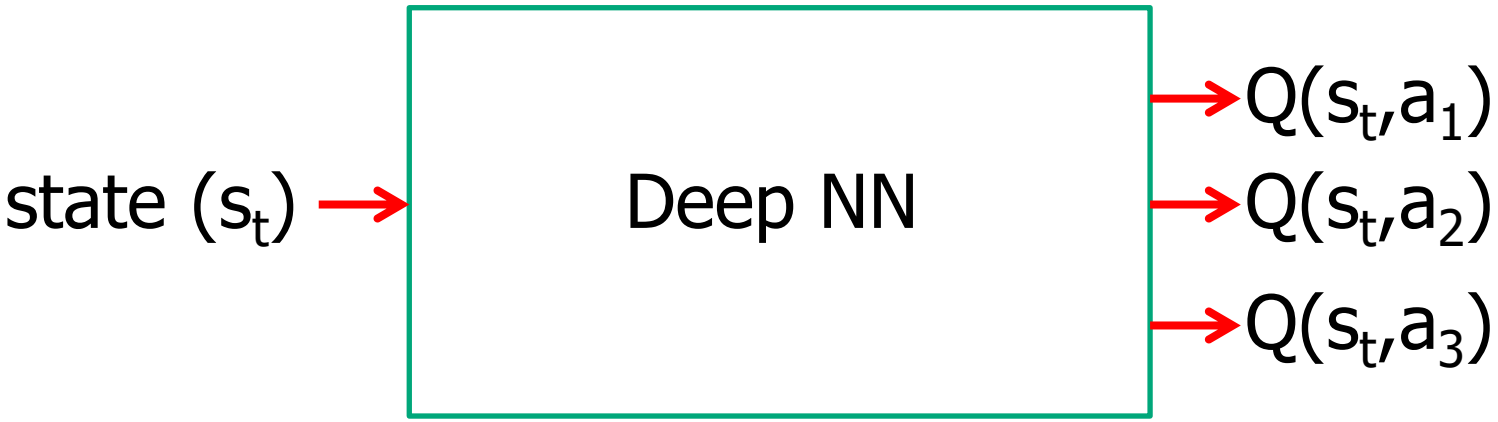
- Utilizarea unei aproximari functionale pentru a estima valoarea Q optima a unei actiuni dintr-o stare
- Utilizeaza o retea CNN care sa indice cea mai buna actiune de efectuat



Deep Q-Networks

- Eroarea retelei este obtinuta prin compararea valorii Q cu valoarea target

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(\underbrace{r_t + \gamma \max_{a'} Q(s_{t+1}, a')}_{\text{target}} - Q(s_t, a_t))$$



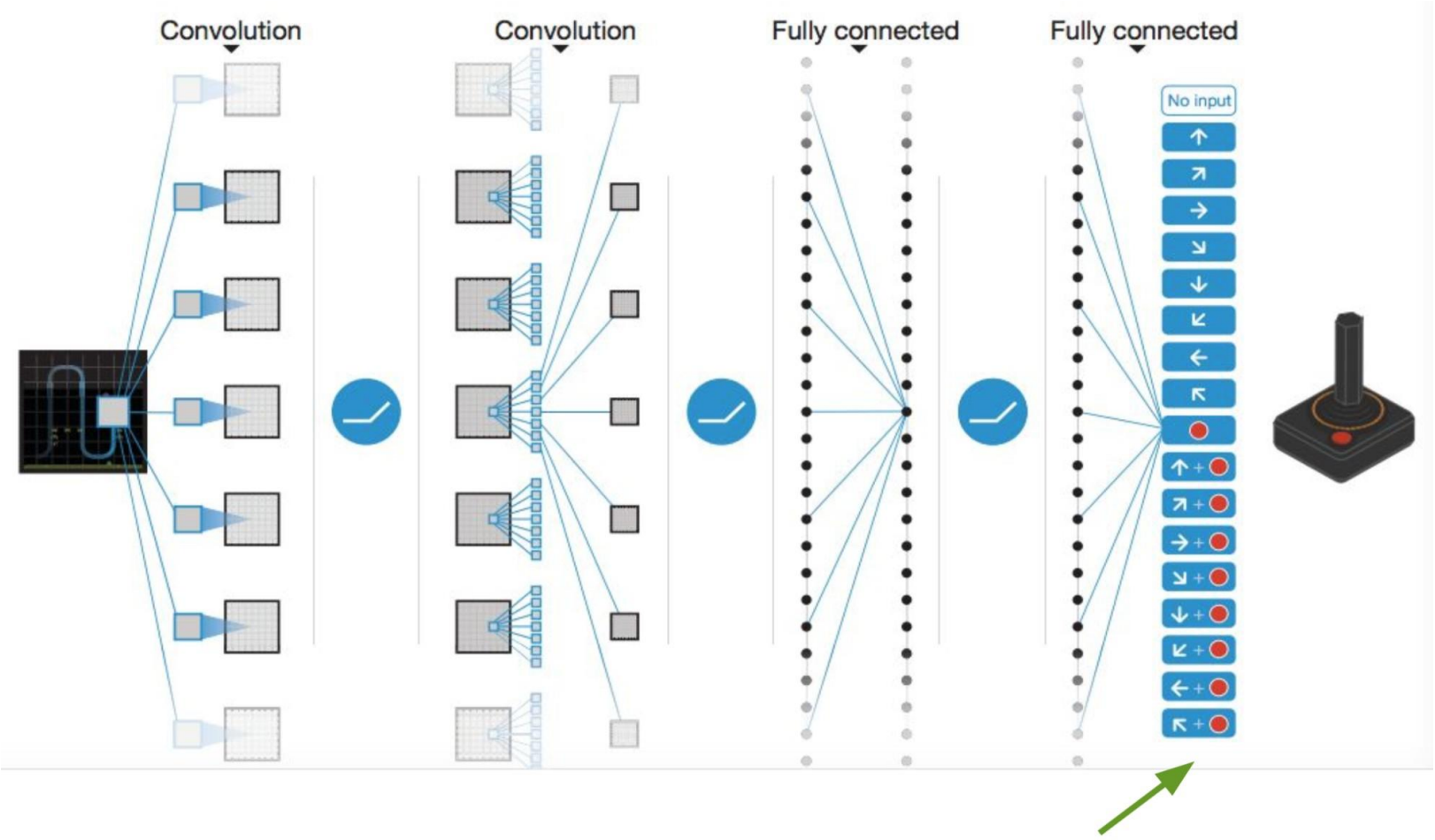
Reprezentare

- 49 de jocuri Atari 2600 folosing Arcade Learning Environment
- Utilizeaza pt toate jocurile aceeasi arhitectura de retea, aceleasi valori de parametrii (dim pas, atenuare, explorare/exploatare, etc.)
- Intrarea este creata numai pe baza imaginilor din jocul video
- Pt fiecare joc invatarea s-a facut prin interactiune cu 50 de milioane de frames

Reprezentare

- Un frame – imagine de 210x160 de pixeli cu 128 culori la 60Hz, transformata intr-o matrice de 84x84 de pixeli (luminozitate)
- O stare reprezentata de 4 frame-uri consecutive (stack 4 most recent frames)
- Intrarea retelei este deci 84 x 84 x 4 - starea ***s***
- Iesirea retelei – valori ***Q(s,a)*** pentru fiecare actiune posibila (miscare joystick), intre 3 si 16
- Recompensa este schimbarea scorului la fiecare pas
- Episoade finite, cate un episode pt fiecare joc jucat, cu momente de timp de la 0 la T
- Recompensa atenuata cu γ

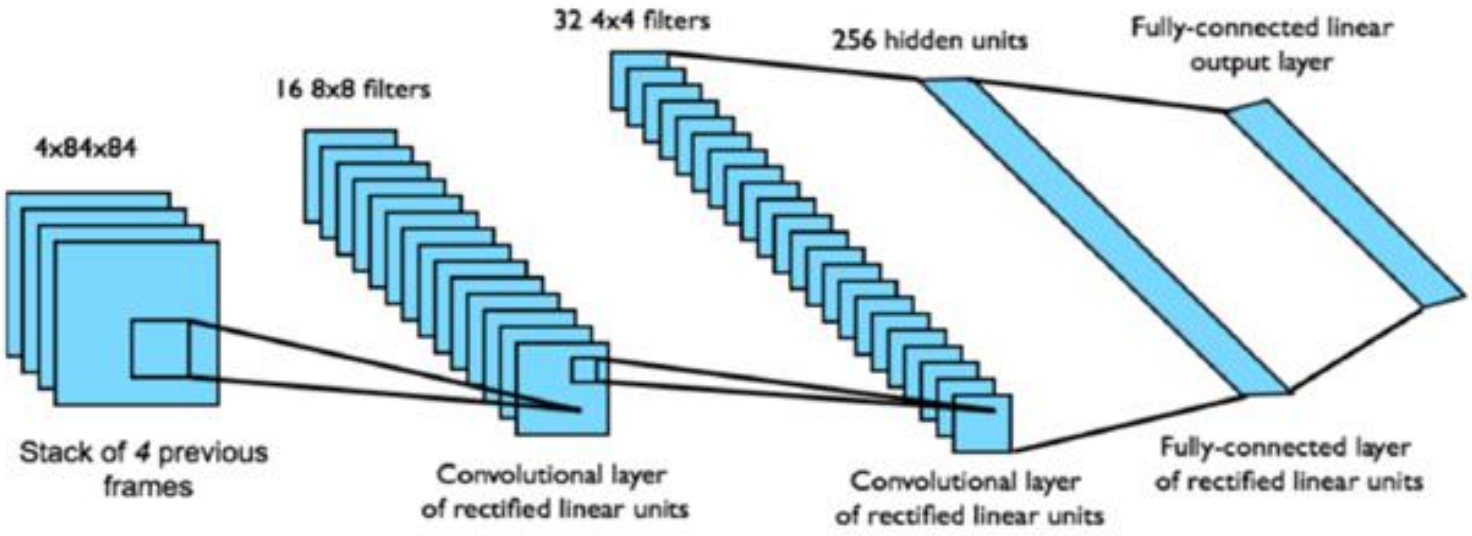
Arhitectura retelei



"*Human-level control through deep reinforcement learning*", Mnih et al, 2015

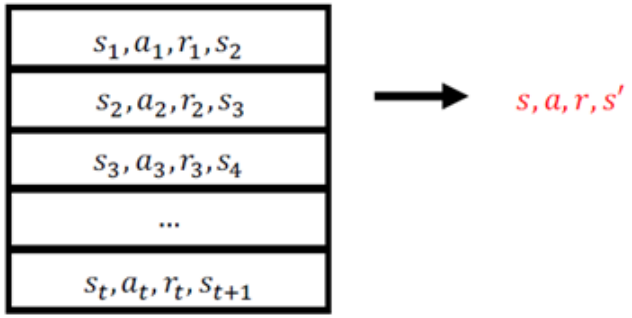
Arhitectura retelei

- Primul strat este de convolutie cu 16 filtre de 8 x 8, stride 4 si ReLU
- Al doilea strat tot de convolutie cu 32 de filtre 4 x 4, stride 2 si ReLU
- Un strat ascuns fully-connected de 256 ReLU
- Stratul de iesire – fully connected cu 1 iesire pt fiecare actiune valida intr-un joc



Functionare

- Utilizeaza Memory replay buffer (s, a, r, s')



- Utilizeaza MSE si SGD

$$J(\mathbf{w}) = \mathbb{E}_{(s_t, a_t, r_t, s_{t+1})}[(\underbrace{y_t^{DQN}}_{\text{target}} - \hat{q}(s_t, a_t, \mathbf{w}))^2]$$

$$y_t^{DQN} = r_t + \gamma \max_{a'} \hat{q}(s_{t+1}, a', \mathbf{w}^-)$$

Instabilitate in Q-Leraning

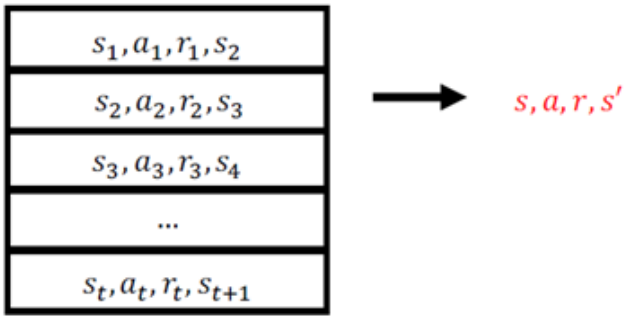
- Q-learning este instabil sau chiar divergent daca se utilizeaza un VFA neliniar pt a reprezenta fct Q sau poate ajunge in minime locale
- Aceasta instabilitate are mai multe cauze:
  - corelarea in secventa de frames (observatii)
  - actualizari mici ale valorii Q pot schimba semnificativ politica si deci schimba distributia datelor
  - corelarea intre valorile actiunilor si valorile target
- pentru a le elimina/minimiza, DQN foloseste
  - Experience replay (replay buffer)
  - 2 retele neurale

Experience replay

- Esantioanele de comportare successive (s, a, r, s')

sunt intens corelate

- Pentru a elimina aceasta corelare, se tin minte "snapshots" ale jocului in replay buffer (dim aprox 1 mil) din care se aleg stochastic minibatch-uri pe care se aplica GD
- In acest fel se elimina corelarea



Q targets fixe

- Mentine fixe ponderile target-ului pentru mai multe iteratii

$$y_t^{DQN} = r_t + \gamma \max_{a'} \hat{q}(s_{t+1}, a', \mathbf{w}^-)$$

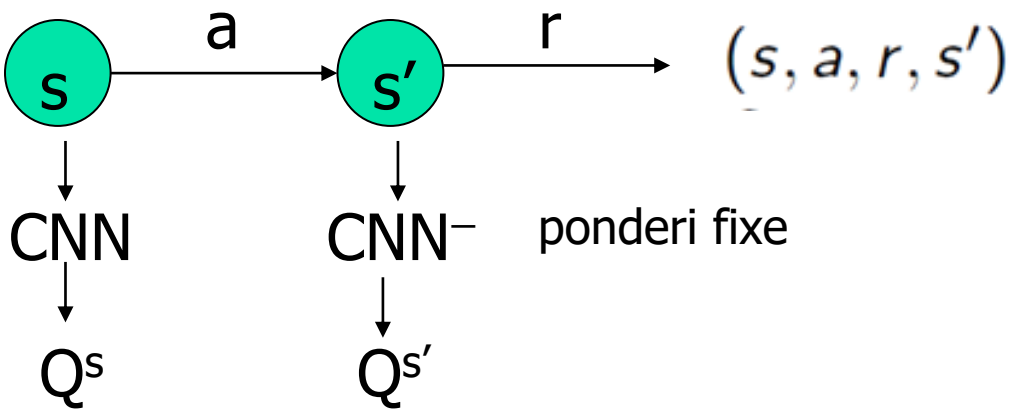
2 retele CNN si CNN$^-$

Reteaua target utilizeaza un set diferit de ponderi fata de cele care sunt actualizate

2 randuri de parametrii

w^- - parametrii utilizati in target

w - parametrii care se actualizeaza



Minimizare J ($Q^s \rightarrow r + Q^{s'}$)

Functionare

$$J(\mathbf{w}) = \mathbb{E}_{(s_t, a_t, r_t, s_{t+1})}[(y_t^{DQN} - \hat{q}(s_t, a_t, \mathbf{w}))^2]$$

$$y_t^{DQN} = r_t + \gamma \max_{a'} \hat{q}(s_{t+1}, a', \mathbf{w}^-)$$

- Calculeaza valorile target
  - Obtine un esantion din buffer $(s, a, r, s') \sim \mathcal{D}$:
  - calculeaza valoarea target pe baza acestui esantion

$$r + \gamma \max_{a'} \hat{Q}(s', a'; \boldsymbol{w}^-)$$

- Foloseste SGD pentru actualizarea ponderilor retelei

$$\Delta \boldsymbol{w} = \alpha(r + \gamma \max_{a'} \hat{Q}(s', a'; \boldsymbol{w}^-) - \hat{Q}(s, a; \boldsymbol{w}))\nabla_{\boldsymbol{w}} \hat{Q}(s, a; \boldsymbol{w})$$

Functionare

- Pasteaza ponderile target w^- fixe pentru mai multi pasi (de ex 50, 100) pentru a imbunatati stabilitatea
- Acest lucru face target mai stabila si reduce zgomotul din target

Ponderile din target w^- sunt actualizate din cand in la cele din reteaua Q - w

$$w^- = w$$

Deep Q-Learning DQN

Initializeaza D - replay memory - cu o capacitate fixa

Initializeaza functia $\hat{Q}$ cu valori aleatoare ale ponderilor w

Initializeaza functia target $\hat{Q}$ cu valori $w^- = w$

pentru episod=1,M **repeta**

observa imaginea initiala x_1 si preproceseaza pentru a obtine s_1

pentru t=1,T **repeta**

selecteaza o actiune $a_t = \begin{cases} \text{aleator cu probabilitatea } \varepsilon \\ argmax_a \hat{Q}(s,a,w) \text{ cu prob 1- } \varepsilon \end{cases}$

executa a_t in emulator si obtine recompensa r_t si imaginea x_{t+1}

preproceseaza s_t, x_{t+1} pentru a obtine s_{t+1} si memoreaza (s_t, a_t, r_t, s_{t+1}) in D

selecteaza uniform un minibatch de N elem (s_j, a_j, r_j, s_{j+1}) din D

daca episodul se termina la pasul j+1 **atunci** $y_j = r_j$

altfel $\boldsymbol{y_j = r_j + \gamma max_{a'} \hat{Q}(s_{j+1}, a', w^-)}$

executa SGD cu eroarea $J(w) = \frac{1}{N}\sum_{j=1}^{N}(y_j - \hat{Q}(s_j, a_j, w))^2$ pentru w

dupa fiecare C pasi actualizeaza $w^- = w$

Imbunatatiri

- Politica de management a buffer-ului

- Double DQNs (2016)
- In DQN standard se folosesc aceleasi valori pentru a selecta si a evalua o actiune.

$$y_t^{DQN} = r_t + \gamma \max_{a'} \hat{q}(s_{t+1}, a', \mathbf{w}^-)$$

- Acest lucru poate conduce la o supraestimare a valorilor
- Pentru a evita acest lucru, se poate decupla selectia actiunii de evaluarea actiunii
- Aceasta este idea din Double DQN

Double DQN

- In DDQN se invata 2 valori de functii prin actualizarea aleatoare a 2 valori de functii la fiecare experienta a.i. se pastreaza 2 seturi de ponderi
- La fiecare actualizare, un set de ponderi este utilizat pentru a determina politica greedy si celalat set de ponderi pentru a determina valoarea.
- Selectarea unei actiuni in argmax este datorata ponderilor **w**, ceea ce insemana ca, la fel ca in Q-Learning, se estimeaza valoarea politicii greedy in functie de valorile curente definite de **w**.
- In schimb se folosesc celelate ponderi $\mathbf{w}^-$ pentru o evaluare corecta a acestei politici.

$$y_t^{DQN} = r_t + \gamma \max_{a'} \hat{q}(s_{t+1}, a', \mathbf{w}^-)$$

$$\Delta \boldsymbol{w} = \alpha(r + \gamma \overbrace{\hat{Q}(\underbrace{\arg\max_{a'} \hat{Q}(s', a'; \boldsymbol{w})}_{\text{Action selection: } \boldsymbol{w}}; \boldsymbol{w}^-)}^{\text{Action evaluation: } \boldsymbol{w}^-} - \hat{Q}(s, a; \boldsymbol{w}))$$

Invatare automata

Universitatea Politehnica Bucuresti
Anul universitar 2021-2022

Adina Magda Florea

Curs 9

Algoritmi genetici

- **Introducere**
- **Schema de baza**
- **Modelare probleme**
- **Exemplu**
- **Selectie**
- **Recombinare**
- **Mutatie**
- **Implementare paralela AG**

1. Introducere

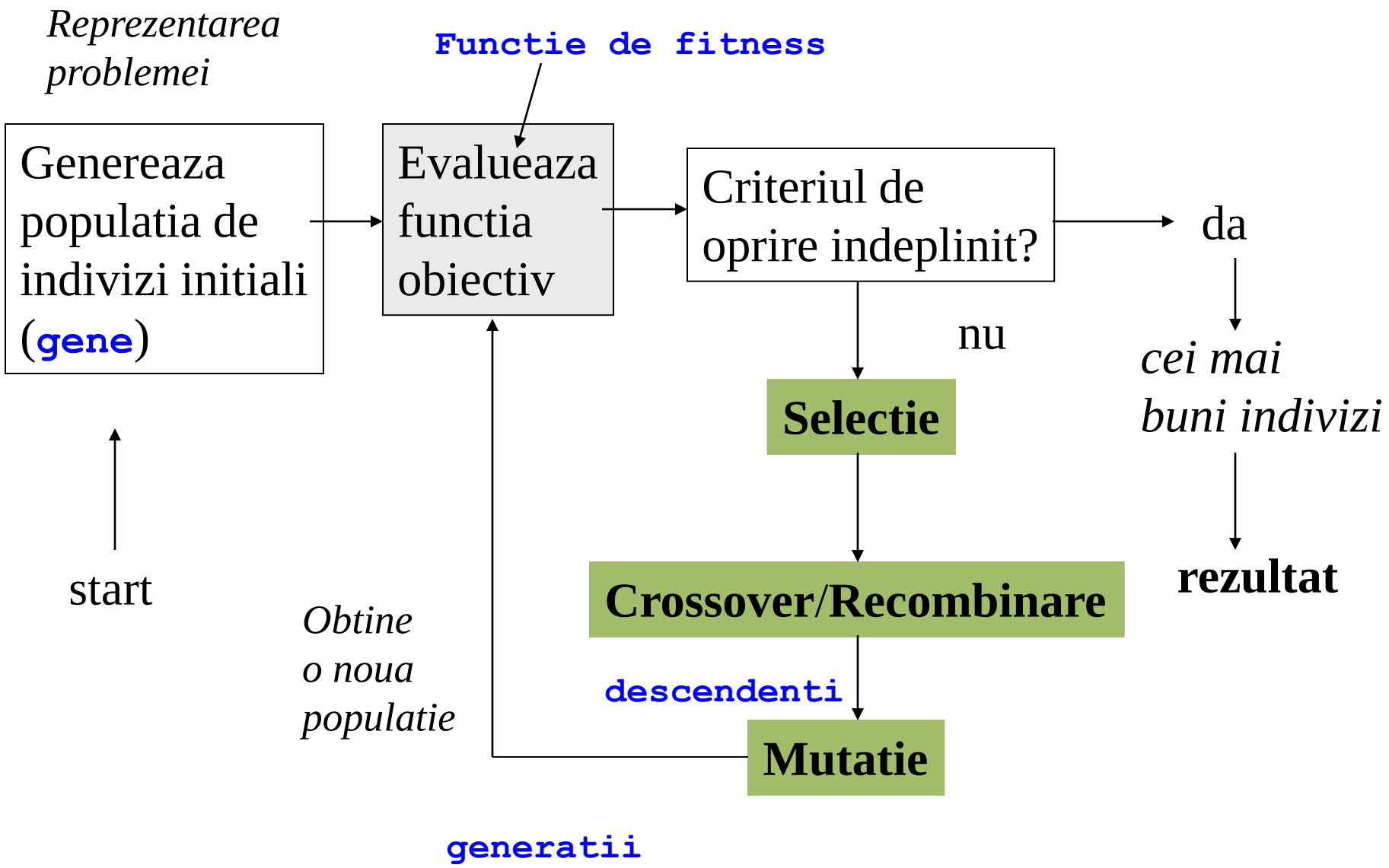
- GA - USA, J. H. Holland (1975)
- Algoritmi evolutivi – Germania, I. Rechenberg, H.-P. Schwefel (1975-1980)
- Programare genetica (2000) J. Koza

- **Optimizare**
- **Modele economice**
- **Modele ecologice**
- **Modele ale sistemelor sociale**
- **Invatare automata**

Introductere - cont

- Opereaza asupra unei populatii de indivizi = solutii potentiale – aplica principiul supravietuirii pe baza de adaptare (fitness)
- Fiecare generatie – o noua aproximatie a solutiei
- Evolutia unei populatii de indivizi mai bine adaptati mediului
- Modeleaza procese naturale: selectie, recombinare, mutatie, migrare, localizare
- Populatie de indivizi – cautare paralela

2. Schema de baza



3. Modelare probleme

Populatie initiala

- Stabileste reprezentare – gena – individ
- Stabileste numar de indivizi in populatie
- Stabileste functia de evaluare (obiectiv)
- Populatia initiala (genele) creata aleator

Selectie

- **Selectie** – extragerea unui subset de gene dintr-o populatie existenta in fct de o definitie a calitatii - **functia de evaluare**
- Determina indivizii selectati pt recombinare si cati **descendenti** (offsprings) produce fiecare individ

Modelare probleme

Criteriul de oprire

- solutie care satisface criteriul
- numar de generatii
- buget
- platou pt cel mai bun fitness

Multipopulation GAs

- Rezultate mai bune - **subpopulatii**
- Fiecare populatie evolueaza separat
- Indivizi sunt schimbati dupa un numar de generatii

3.1 Selectie

(1) Primul pas: atribuire fitness

- atribuire proportionala
- atribuire bazata pe rang

(2) Selectia efectiva: parintii sunt selectati in fct de fitness pe baza unuia din algoritmii:

- **roulette-wheel selection** (selectie ruleta)
- **stochastic universal sampling** (esantionare universala stohastica)
- **local selection** (selectie locala)
- **tournament selection** (selectie turneu)
- **proportional selection** (selectie proportionala)

3.2 Reinserare

- Offspring – daca se produc mai putini indivizi sau mai putini copii atunci indivizi suplimentari trebuie reinserati in noua populatie
- Algoritmul de selectie determina schema de reinserare (in general)

❑ **reinserare globala** – in toata populatia, de ex. pt. roulette-wheel selection, stochastic universal sampling

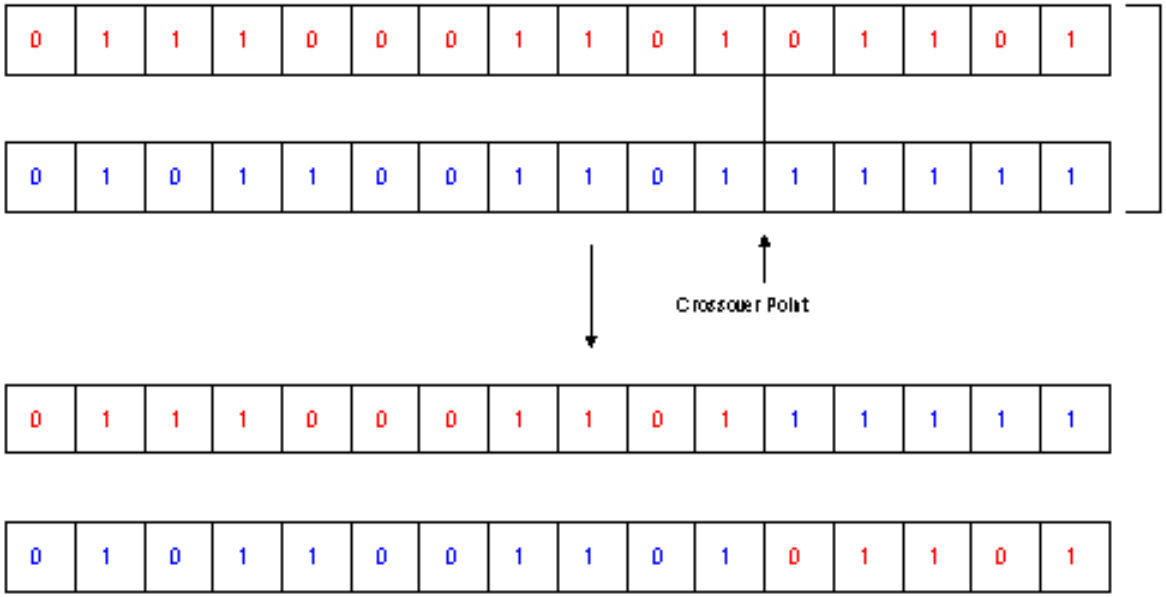
❑ **reinserare locala** pt selectie locala

3.3 Crossover/Recombination

- Recombinarea produce noi indivizi prin combinarea informatiei din parinti (parents - mating population).
- Diverse scheme de recombinare
- ***O posibilitate*** – imperechere aleatoare
- La fel cu ***Crossing Over*** din genetica

1. *Un procent* P_M din indivizii noii populati sunt selectati si se imperecheaza aleator
2. Un *crossover point* este selectat pentru fiecare pereche (acelasi sau diferit cu probabilitate)
3. Informatia este schimbata intre cei doi indivizi pe baza pct de crossover

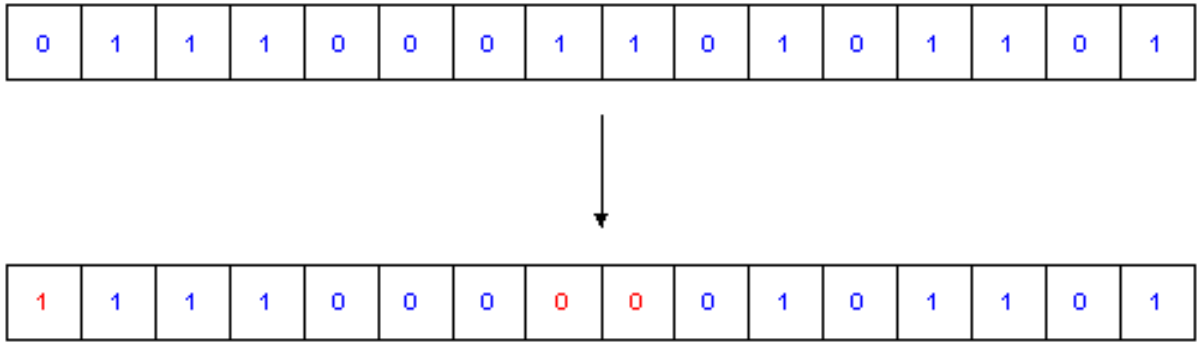
Crossover


0 1 1 1 0 0 0 1 1 0 1 0 1 1 0 1
0 1 0 1 1 0 0 1 1 0 1 1 1 1 1 1
Crossover Point
0 1 1 1 0 0 0 1 1 0 1 1 1 1 1 1
0 1 0 1 1 0 0 1 1 0 1 0 1 1 0 1

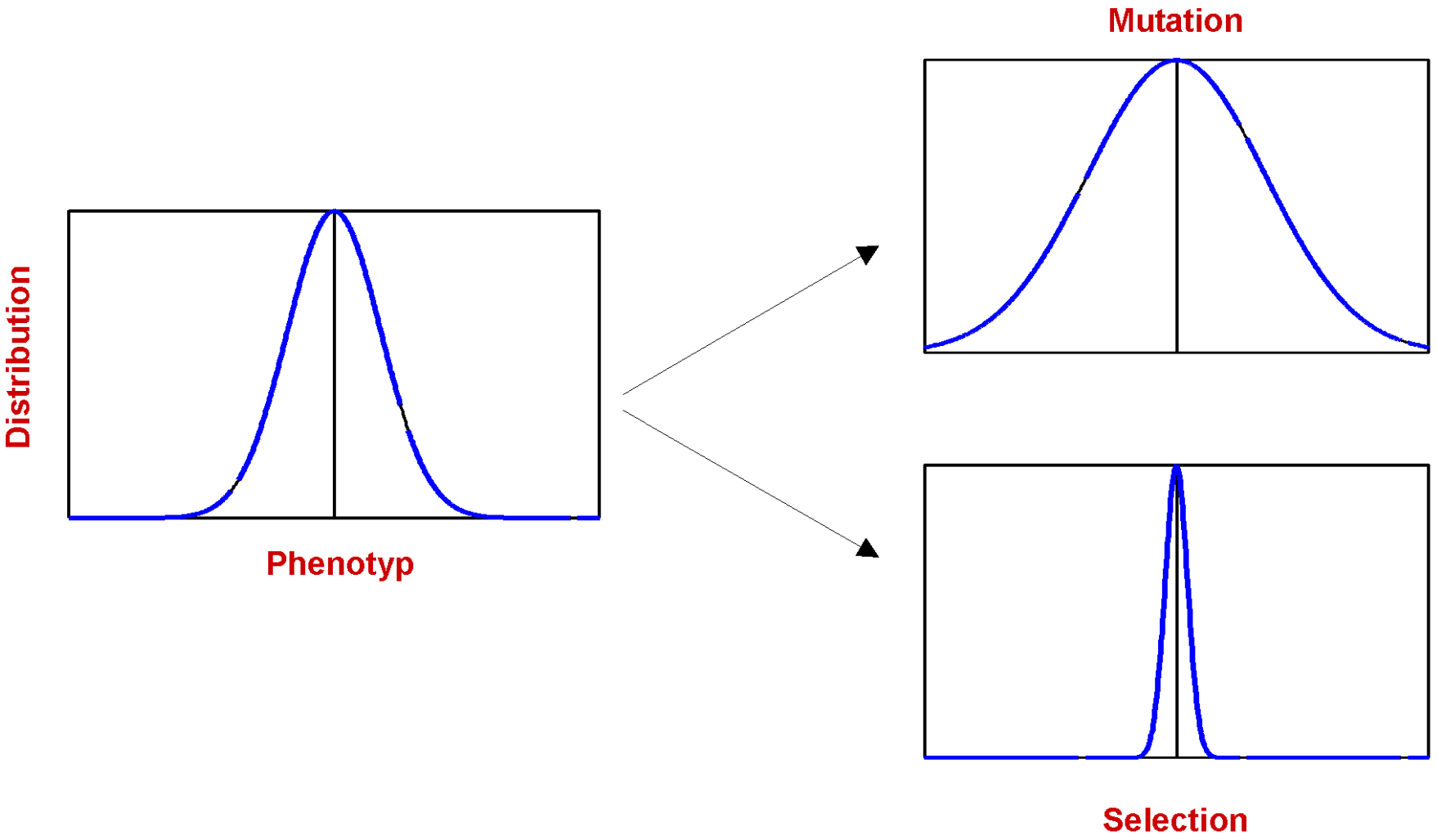
3.4 Mutatie

- Offspring - mutatie
- Mutatie cu perturbatii mici aleatoare
- Diverse forme de mutatie, depind de reprezentare
- **Mutatie** – ***explorare*** *vs exploatare*
- Schema simpla
  - *Fiecare bit are o probabilitate de mutatie*

Mutatie



Efectul mutatiei si a selectiei



4. Selectia in detaliu

- Primul pas este **atribuirea de fitness** (A)
  - Atribuire directa pe baza functiei obiectiv SAU
  - Atribuire pe baza unui mecanism (A1, A2)
- Fiecare individ din populatie primeste o valoare de fitness
- Pe baza valorii de fitness se realizeaza **selectia** (S) dupa o schema de selectie (S1, S2, S3, S4)

A1. Atribuire fitness proportionala

- *Fiecare gena are un fitness asociat*
- Se calculeaza fitness mediu al populatiei
- Fiecare individ va fi copiat in noua populatie in fct de fitness comparat cu fitness mediu
- *fitness mediu 5.76, fitness individ 20.21 – se copiaza de 3 ori*
- Indivizii cu fitness egal sau sub medie se ignora
- Noua populatie – poate fi mai mica
- Noua populatie se completeaza cu indivizi selectati aleator din vechea populatie

A2. Atribuire fitness bazata pe rang

- Fitness-ul atribuit fiecarui individ depinde numai de pozitia lui intre indivizii din populatie
- Pozitia unui individ in populatie depinde de functia obiectiv
- *Pos* = 1 – cel mai putin bun
- *Pos* = *Nind* – cel mai bun
- Populatia este ordonata in functie de fitness

Atribuire fitness bazata pe rang

- *Nind* – numarul de indivizi din populatie
- *Pos* – pozitia individului in populatie (cel mai prost *Pos*=1, cel mai bun *Pos*=*Nind*)
- *SP* – presiunea de selectie - fitness maxim impartit la fitness mediu – indica puterea unui mechanism de selectie

Rang liniar

Fitness(*Pos*) = 2 - *SP* + 2*(*SP* - 1)*(*Pos* - 1) / (*Nind* - 1)

- Rangul liniar permite valori SP in (1.0, 2.0].

Atribuire fitness bazata pe rang

Rang neliniar:

$$\mathbf{Fitness(}\mathit{Pos}\mathbf{) = } \mathit{Nind} * X^{(\mathit{Pos} - 1)} / \Sigma_{i = 1,\mathit{Nind}} (X^{(i - 1)})$$

- X se calculeaza ca radacina a polinomului:

$$0 = (SP - Nind) * X^{(Nind - 1)} + SP * X^{(Nind - 2)} + \ldots + SP * X + SP$$

- Rang neliniar permite valori SP in

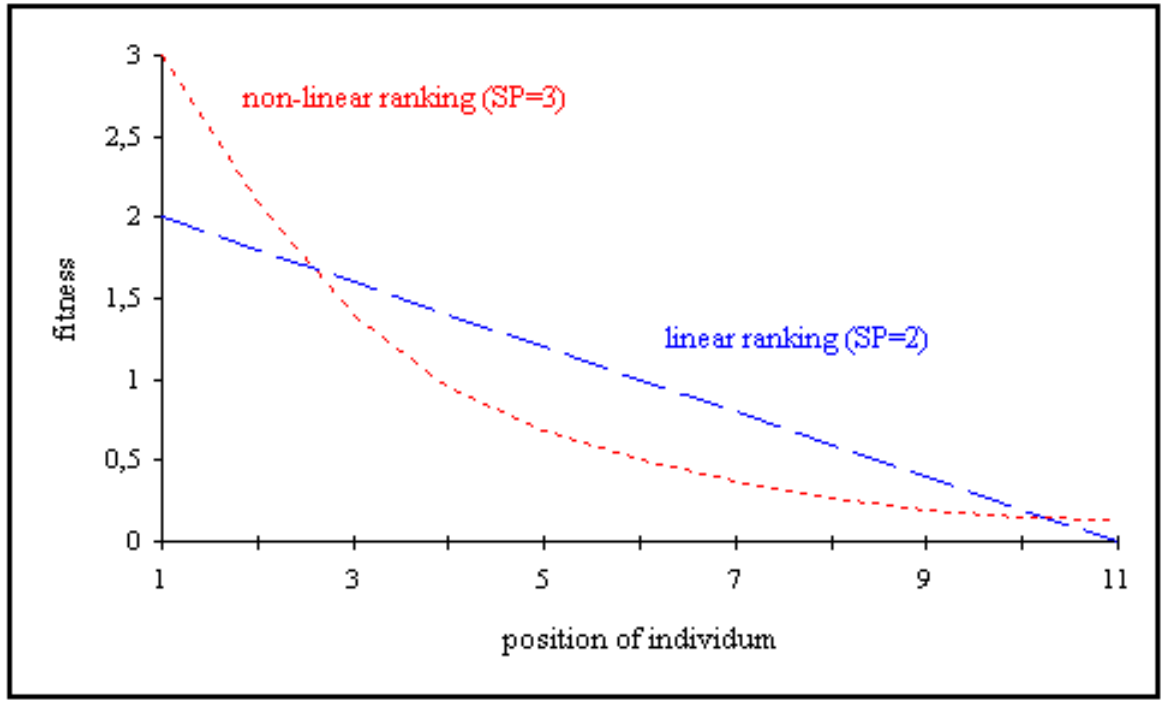
$$[1.0, \mathit{Nind} - 2.0]$$

- SP mai mari

Atribuire fitness bazata pe rang

Atribuire fitness rang liniar si rang neliniar

Liniar
SP=2 $\rightarrow$ 0..2
SP = 1.5 $\rightarrow$ 0.5..1.5



Atribuire fitness bazata pe rang

- Fata de atribuirea proportionala:
  - Evita problema stagnarii in cazul in care presiunea de selectie este prea mica sau convergenta prematura genereaza o zona de cautare prea ingusta
  - Ofera un mod simplu de a controla presiunea de selectie
  - In general mai robusta
- **intensitatea de selectie**
- **pierderea diversitatii**

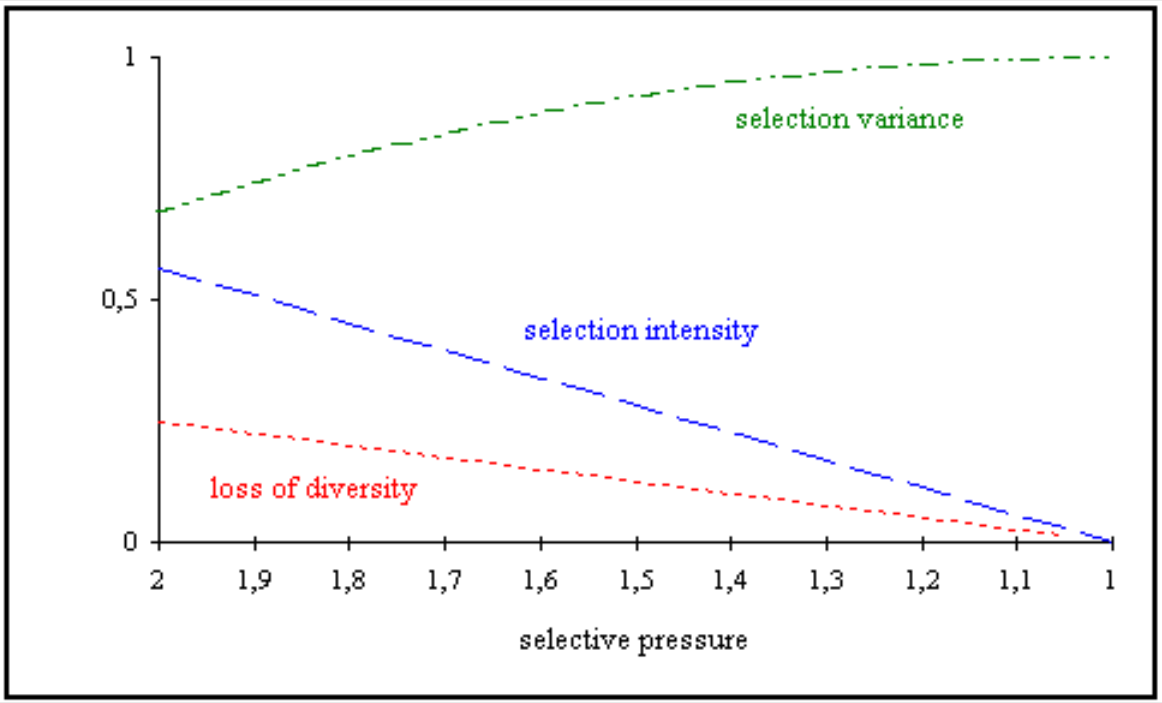
Atribuire fitness bazata pe rang

Proprietati

Intensitatea de selectie: $SelInt_{Rank}(SP) = (SP-1)*(1/sqrt(\pi))$.

Pierderea diversitatii: $LossDiv_{Rank}(SP) = (SP-1)/4$.

Covarianta selectiei: $SelVar_{Rank}(SP) = 1-((SP-1)^2/ \pi) = 1-SelInt_{Rank}(SP)^2$.



S1. Selectia bazata pe ruleta

"Roulette-wheel selection" sau "stochastic sampling with replacement"

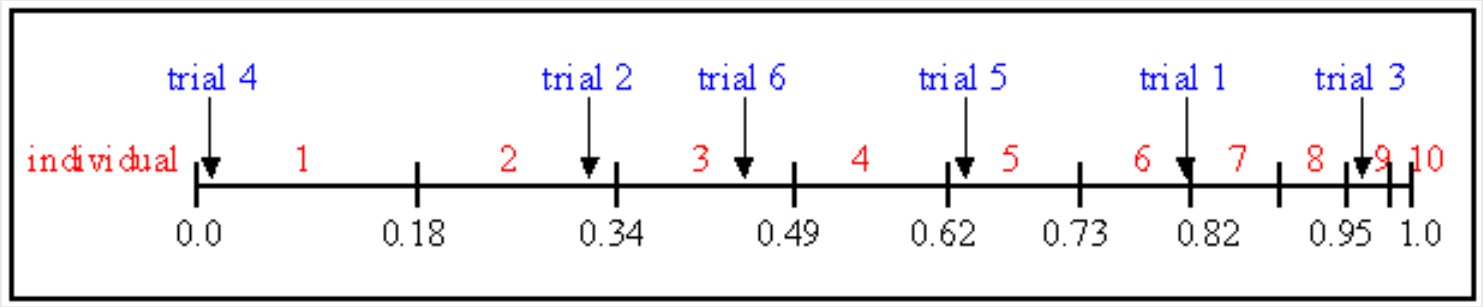
11 indivizi, rang liniar si SP = 2

Nr individ	1	2	3	4	5	6	7	8	9	10	11
Valoare fitness	2.0	1.8	1.6	1.4	1.2	1.0	0.8	0.6	0.4	0.2	0.0
Probabil. de selectie	0.18	0.16	0.15	0.13	0.11	0.09	0.07	0.06	0.03	0.02	0.0

6 numere aleatoare (distribuite uniform intre 0.0 si 1.0):

- 0.81, 0.32, 0.96, 0.01, 0.65, 0.42.

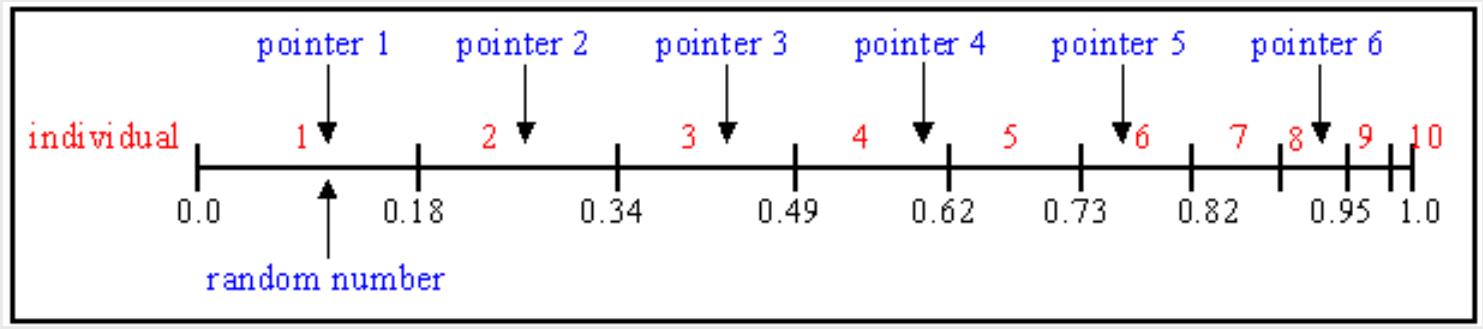
6, 2, 9, 1, 5, 3



S2. Esantionare universala stohastica

- Se plaseaza pointeri la distante egale pe un interval [0..1] – atatia pointeri cati indivizi se vor selecta
- *NPointer* – numarul de indivizi care va fi selectat
- Distanta intre pointeri 1/*Npointer*
- Pozitia primului pointer este data de un numar aleator in intervalul [0..1/*NPointer*].
- Pentru 6 indivizi de selectat, distanta intre pointeri este 1/6=0.167.
- **1 numar aleator in intervalul [0, 0.167]:** 0.1.

1, 2, 3, 4, 6, 8

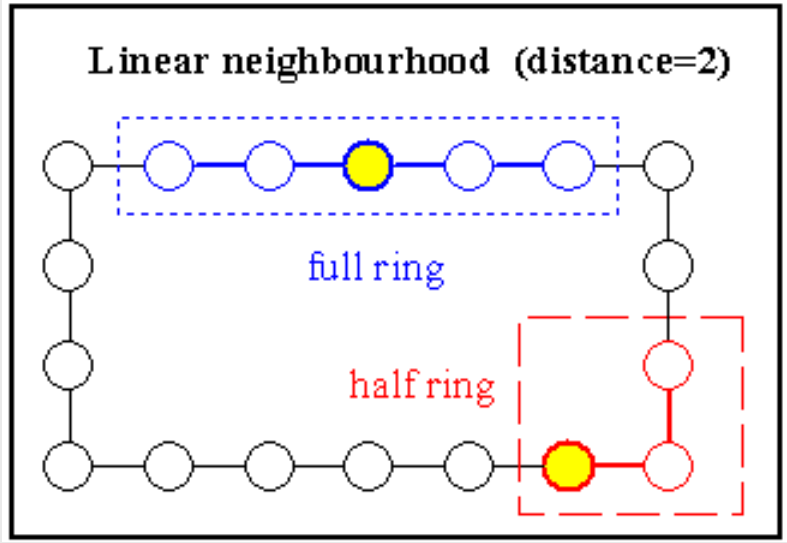


S3. Selectie locala

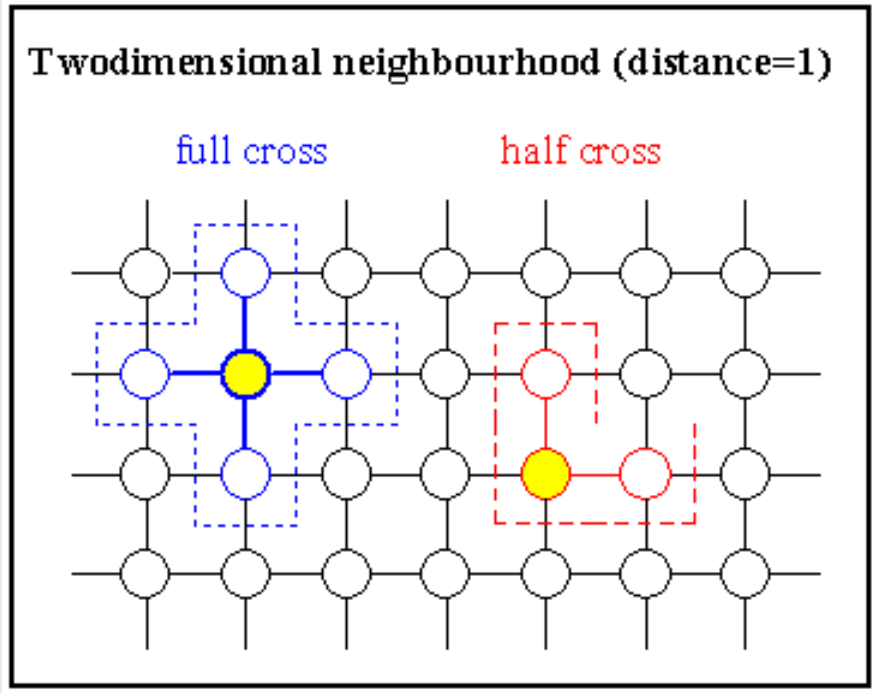
- Se selecteaza prima jumatate a populatiei aleator (sau utilizand un algoritm de selectie – esantionare stohastica sau turneu).
- Se defineste apoi o vecintate pentru fiecare individ selectat.
- Se selecteaza un alt individ pt recombinare din vecintate (best, fitness proportional, sau aleator).

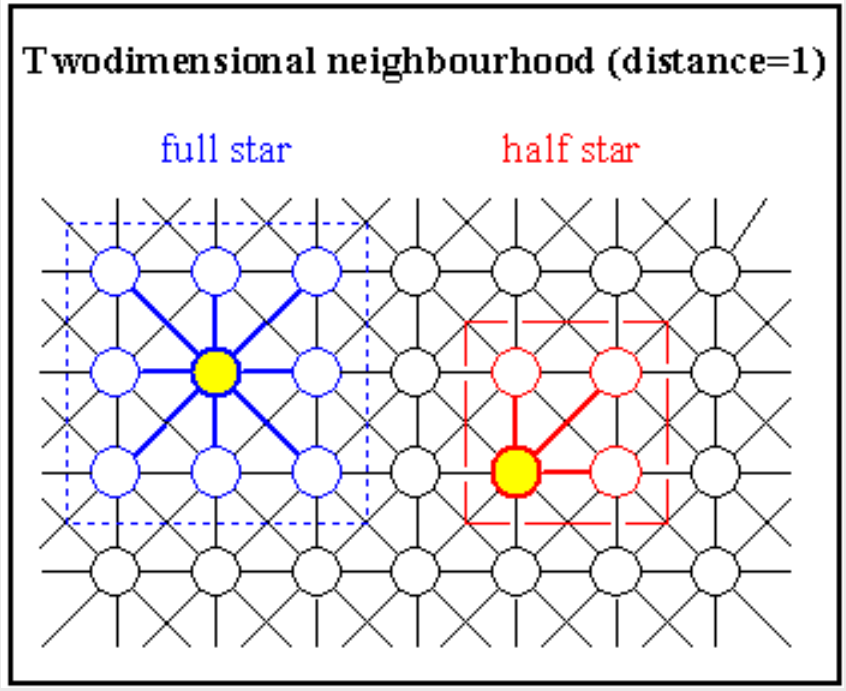
Selectie locala

- Fiecare individ este intr-o vecinatate
- Structurarea populatiei
- Vecintatea – grup de indivizi care se pot recombina (potential)
- Vecintatea liniara: full si half ring



Selectie locala





Selectie locala

- Distanta de izolare intre indivizi
- Cu cat mai mica vecintatea, cu atat mai mare izolarea
- Vecintati care se suprapun – apare transmisie de caracteristici
- Dimensiunea vecinatatii determina viteza de propagare
- Propagare rapida vs mentinere diversitate mare
- Diversitate mare – previne convergenta prematura

S4. Selectie turneu

- Un numar *Tour* de indivizi din populatie este selectat aleator si cel mai bun individ dintre acestia este selectat ca parinte
- Procesul se repeta pt cati indivizi dorim sa selectam
- Parametrul pt dimensiunea turneului este *Tour*.
- *Tour* - valori intre 2 .. *Nind*
- Relatie intre *Tour* si intensitatea de selectie

Dimensiune turne	1	2	3	5	10	30
Intensitate selectie	0	0.56	0.85	1.15	1.53	2.04

Estimeaza valoare medie a fitness-ului populatiei dupa aplicarea unei metode de selectie

Selectie turneu

- **Intensitatea de selectie:**

  $SelInt_{Tour}(Tour) = sqrt(2*(log(Tour)-log(sqrt(4.14*log(Tour)))))$

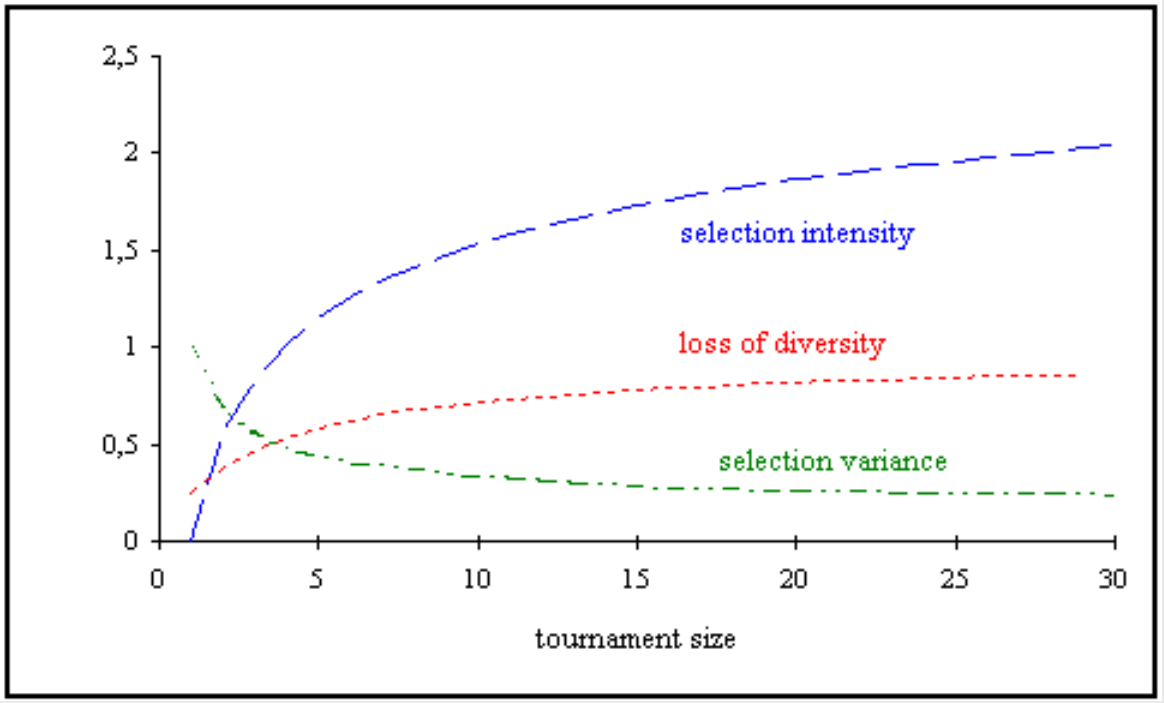
- **Pierderea diversitatii:**

  $LossDiv_{Tour}(Tour) = Tour^{-(1/(Tour-1))} - Tour^{-(Tour/(Tour-1))}$

  (Aprox 50% din populatie se pierde pt Tour=5).

- **Covarianta selectiei:**

  $SelVar_{Tour}(Tour) = 1- 0.096*log(1+7.11*(Tour-1))$, $SelVar_{Tour}(2) = 1-1/\pi$

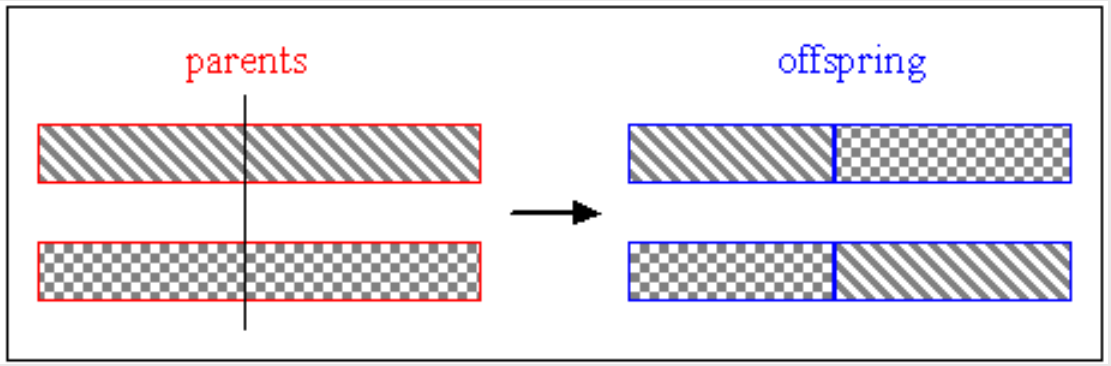


6. Crossover / recombination

- Produce noi indivizi prin recombinarea informatei din parinti
- Reprezentari binare
  - binara
  - multipunct
  - uniforma
- Reprezentari intregi/reale
  - discreta
  - reala intermediara
  - liniara

6.1 Recombinare binara

- Pozitia de crossover selectata aleator ➔ se produc 2 descendenti



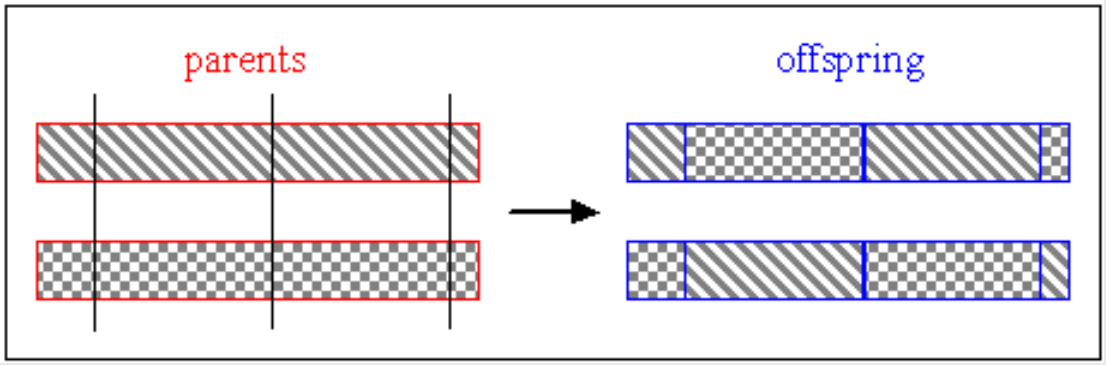
Recombinare binara

- Exemplu
- **individual 1:** 0 1 1 1 0 | 0 1 1 0 1 0
- **individual 2:** 1 0 1 0 1 | 1 0 0 1 0 1

- pozitie crossover = **5**
- Se creaza 2 indivizi noi:
- **offspring 1:** 0 1 1 1 0 | 1 0 0 1 0 1
- **offspring 2:** 1 0 1 0 1 | 0 1 1 0 1 0

6.2 Recombinare multi-punct

- *m* pozitii de crossover k_i



- ***individual 1:*** 0 1 1 1 0 0 1 1 0 1 0
- ***individual 2:*** 1 0 1 0 1 1 0 0 1 0 1
- pos. cross (m=3) 2 6 10
- **offspring 1:** 0 1| 1 0 1 1| 0 1 1 1| 1
- **offspring 2:** 1 0| 1 1 0 0| 0 0 1 0| 0

6.3 Recombinare uniforma

- Generalizeaza binara simpla si multipunct
- **Crossover mask** – aceeasi dimensiune cu a individului;
- creata aleator si paritatea bitilor din masca indica ce parinte va oferi descendentilor bitul respectiv
- *individual 1:* 0 1 1 1 0 0 1 1 0 1 0
- *individual 2:* 1 0 1 0 1 1 0 0 1 0 1

  mask 1: 0 1 1 0 0 0 1 1 0 1 0

  mask 2: 1 0 0 1 1 1 0 0 1 0 1 (inversa a mask 1)

  In masca 1 – individ 1, 0 – individ 2
- *offspring 1:* 1 1 1 0 1 1 1 1 1 1 1
- *offspring 2:* 0 0 1 1 0 0 0 0 0 0 0
- Spears and De Jong (1991) – parametrizare prin asocierea unei probabilitati

6.4 Recombinare reala discreta

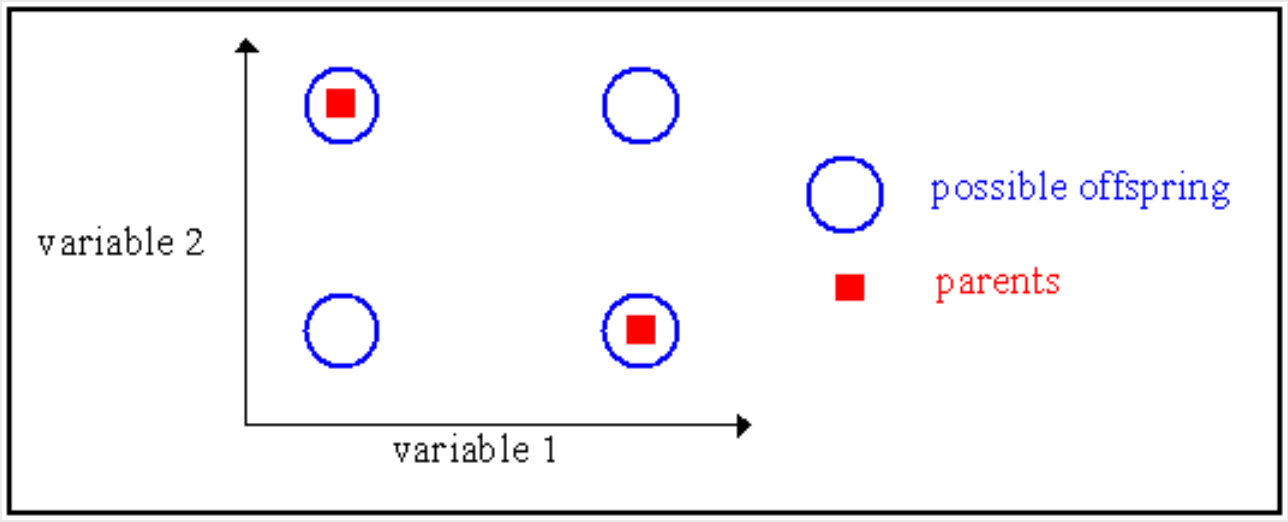
Recombinare discreta

- Schimb de valori reale intre indivizi.
- individual 1: 12 25 5
- individual 2: 123 4 34
- Pt fiecare valoare, parintele care contribuie este ales aleator cu probabilitati egale
- sample 1: 2 2 1
- sample 2: 1 2 1
- Dupa recombinare:
- offspring 1: 123 4 5
- offspring 2: 12 4 5

Recombinare reala discreta

Recombinare discreta

- Pozitiile posibile ale descendentilor



- Poate fi utilizata cu orice valori (binare, reale or simboluri).

Recombinare reala intermediara

Recombinare reala intermediara

- Numai pt valori reale
- Valorile din descendenti alese in jurul valorilor din parinti

- **Regula:**

  offspring = parent 1 + Alpha (parent 2 - parent 1)

  unde Alpha este un factor de scalare ales aleator in intervalul [-*d*, 1 + *d*].

- *d* = 0 sau *d* > 0. O valoare buna d = 0.25.
- Fiecare valoare din descendenti este rezultatul combinarii parintilor cu o noua Alpha pt fiecare variabila

Recombinare reala intermediara

Recombinare reala intermediara

- individual 1: 12 25 5
- individual 2: 123 4 34

Valorile Alpha sunt:

- sample 1: 0.5 1.1 -0.1
- sample 2: 0.1 0.8 0.5
- Indivizi noi

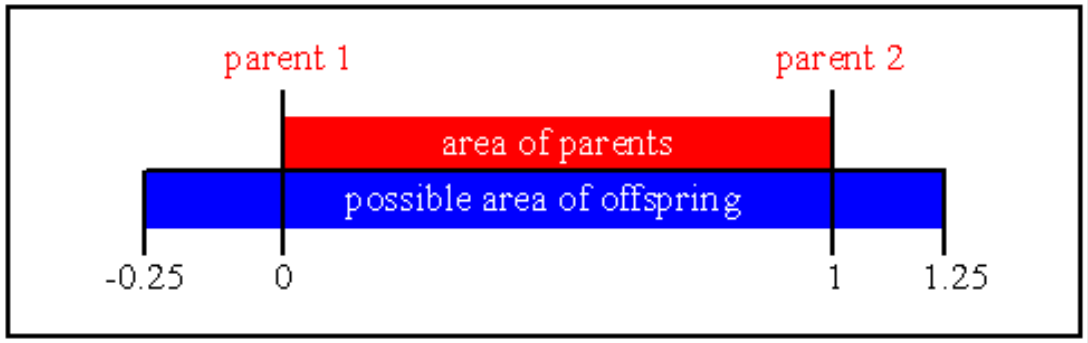
offspring = parent 1 + Alpha (parent 2 - parent 1)

- offspring 1: 67.5 1.9 2.1
- offspring 2: 23.1 8.2 19.5

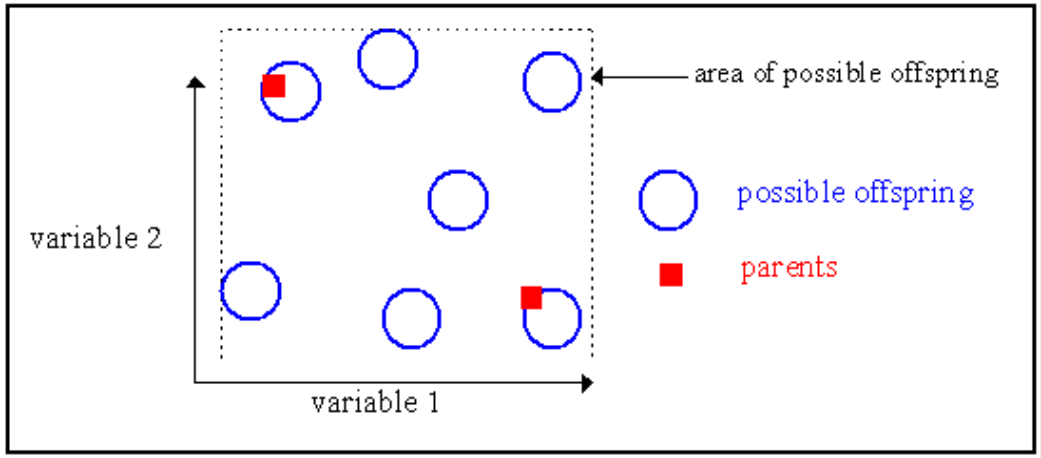
Recombinare reala intermediara

Recombinare reala intermediara

- Domeniul de valori ale descendentilor fata de cel al parintilor



- Repartizare descendenti dupa recombinare intermediara



Recombinare reala liniara

Recombinare liniara

- Similara cu cea intermediara dar se foloseste un singur Alpha.

offspring = parent 1 + Alpha (parent 2 - parent 1)

- individual 1: 12 25 5
- individual 2: 123 4 34

Valorile Alpha sunt:

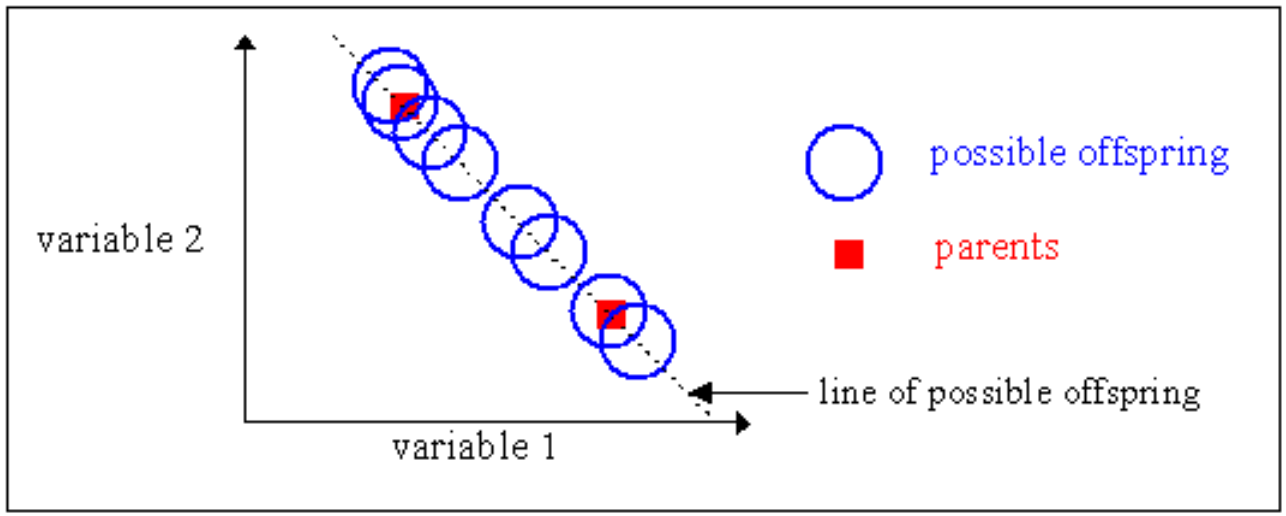
- sample 1: 0.5
- sample 2: 0.1

Indivizii noi:

- offspring 1: 67.5 14.5 19.5
- offspring 2: 23.1 22.9 7.9

Recombinare reala liniara

Recombinare liniara



7. Mutatie

- Dupa recombinare – mutatia descendentilor
- Valori din descendenti sunt mutati prin inversiune (binar) sau adaugarea unor valori mici aleatoare (pasul de mutatie), cu probabilitati mici
- Probabilitatea de mutatie este invers proportionala cu dimensiunea indivizilor
- Cu cat avem indivizi mai lungi cu atat este mai mica probabilitatea de mutatie

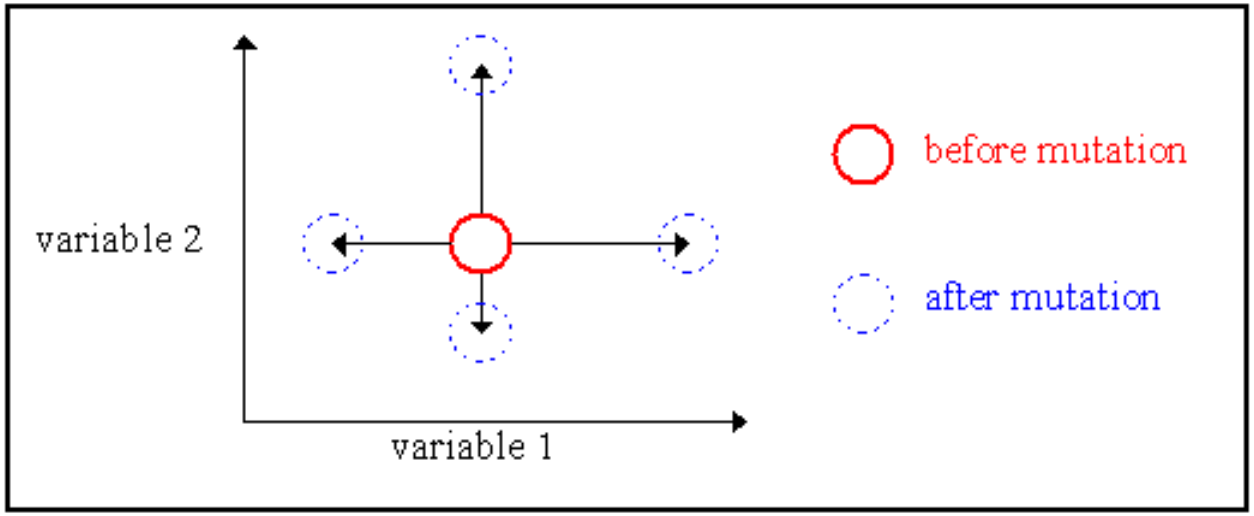
7.1 Mutatie binara

- Schimb valorile binar
- Pt fiecare individ, bitul de mutat este ales aleator
- 11 valori, bit 4

inainte de mutatie	0	1	1	1	0	0	1	1	0	1	0
dupa mutatie	0	1	1	0	0	0	1	1	0	1	0

7.2 Mutatie cu valori reale

- **Efect**
- **mutatie**



- Dimensiune pas – dificil; poate varia in timpul evolutiei
- Mici – bine, lent; mari – mai repede
- Operator mutatie :

  mutated variable = variable ± *range*·*delta* (+ sau – cu probabililate egala)

  range = 0.5*domeniu variabila

  delta = sum($a(i)\ 2^{-i}$), $a(i) = 1$ cu probabilitate $1/m$, altfel $a(i) = 0$.

8 Implementarea paralela a AG

- **Modelul migrator**
- **Model global - worker/farmer**

8.1 Migrare

- Modelul migrator imparte populatia in **subpopulatii**.
- Aceste populatii evolueaza independent un numar de generatii (timp de izolare)
- Dupa timpul de izolare, un numar de indivizi este distribuit intre subpopulatii = **migrare**.
- Diversitatea genetica a subpopulatiilor si schimbul de informatii intre subpopulatii depinde de:

  - numarul de indivizi schimbati = **rata migrare**

  - metoda de **selectie** a indivizilor pentru migrare

  - **schema** de migrare

Migrare

- Implementarea a modelului migrator:
  - scade timpul de prelucrare
  - necesita mai putine evaluari de functii obiectiv fata de un model cu o singura populatie

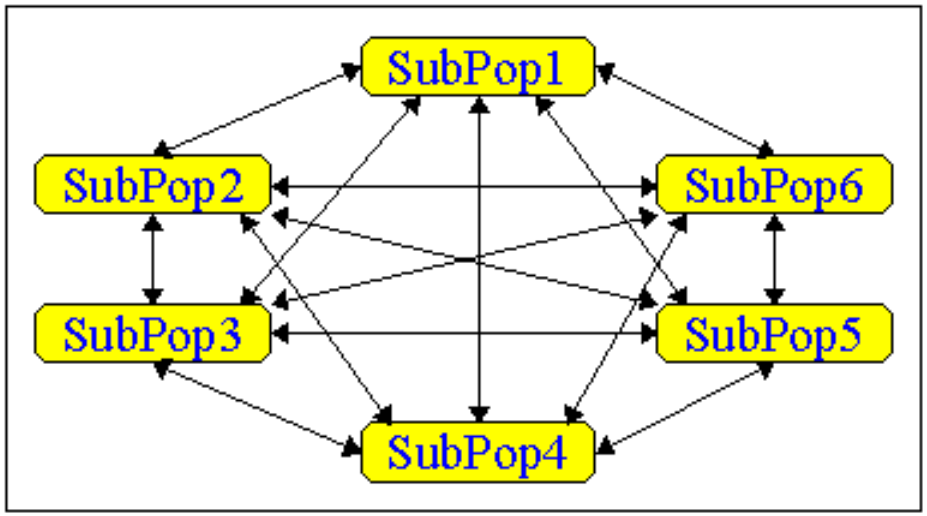
- Implementarea migrarii (paralel) pe un singur prcesor (pseudo-paralel) este buna
- In anumite cazuri se obtin rezultate mai bune

Migrare

- **Selectia** indivizilor pentru migrare:
  - aleator
  - bazata pe fitness (selectez pentru migrare cei mai buni indivizi).
- Schema de migrare intre subpopulatii:
  - intre toate subpopulatiile (topologie completa)
  - topologie inel
  - topologie vecinatate

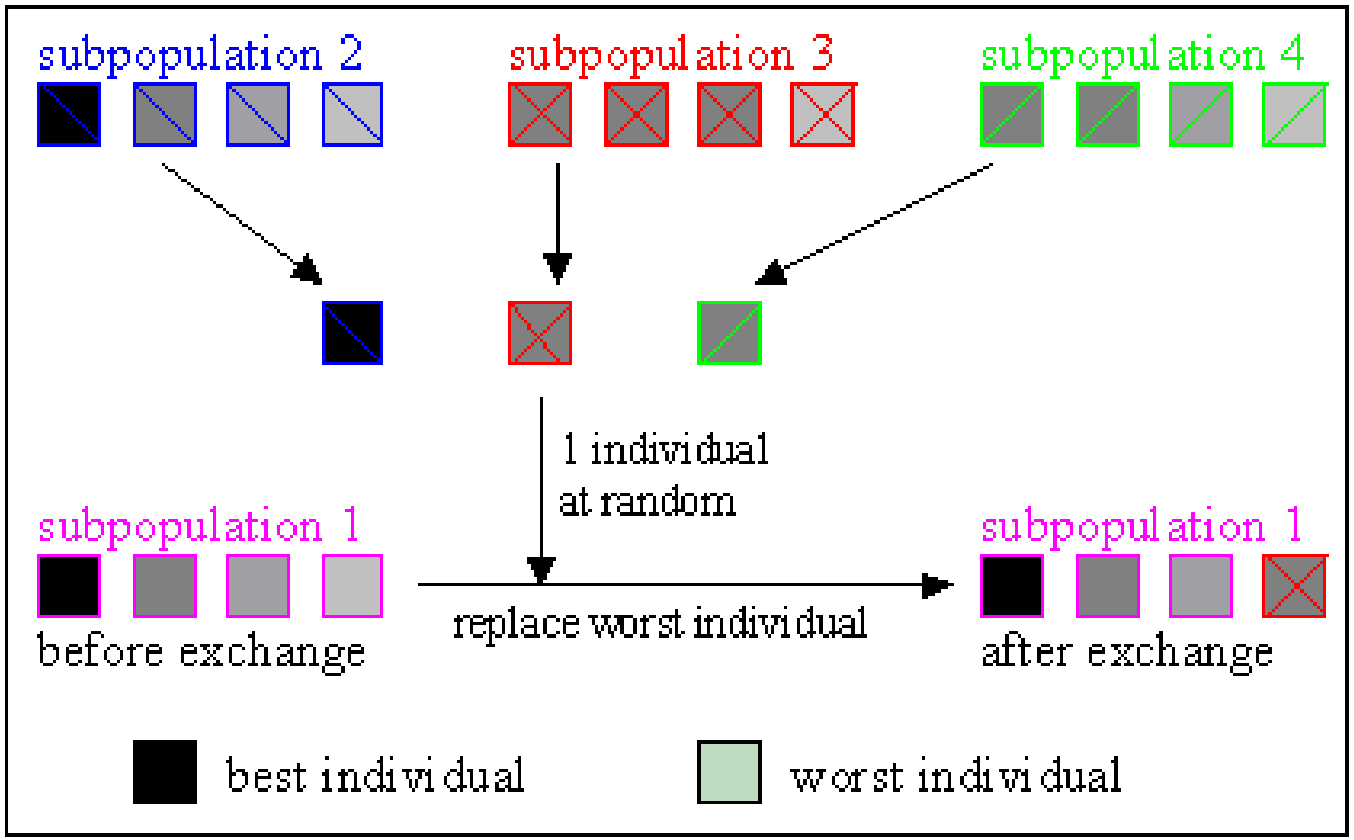
Migrare

Topologie completa



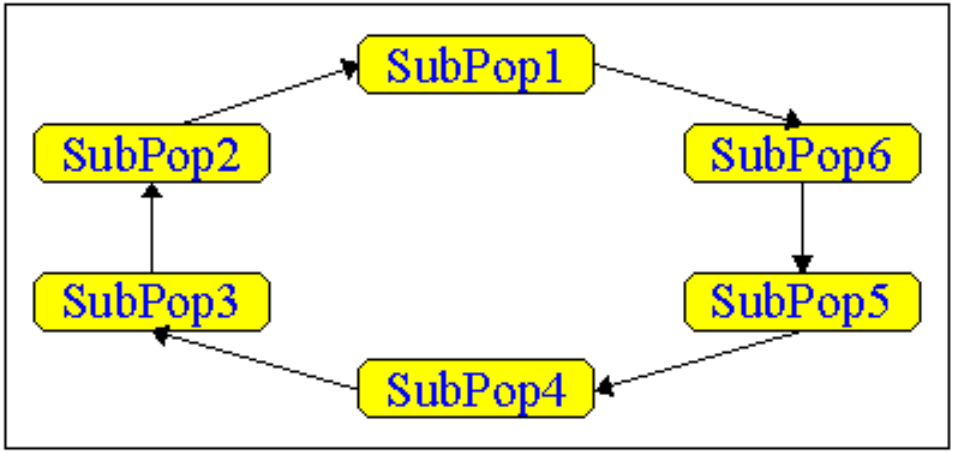
Migrare

Schema de migrare intre subpopulatii

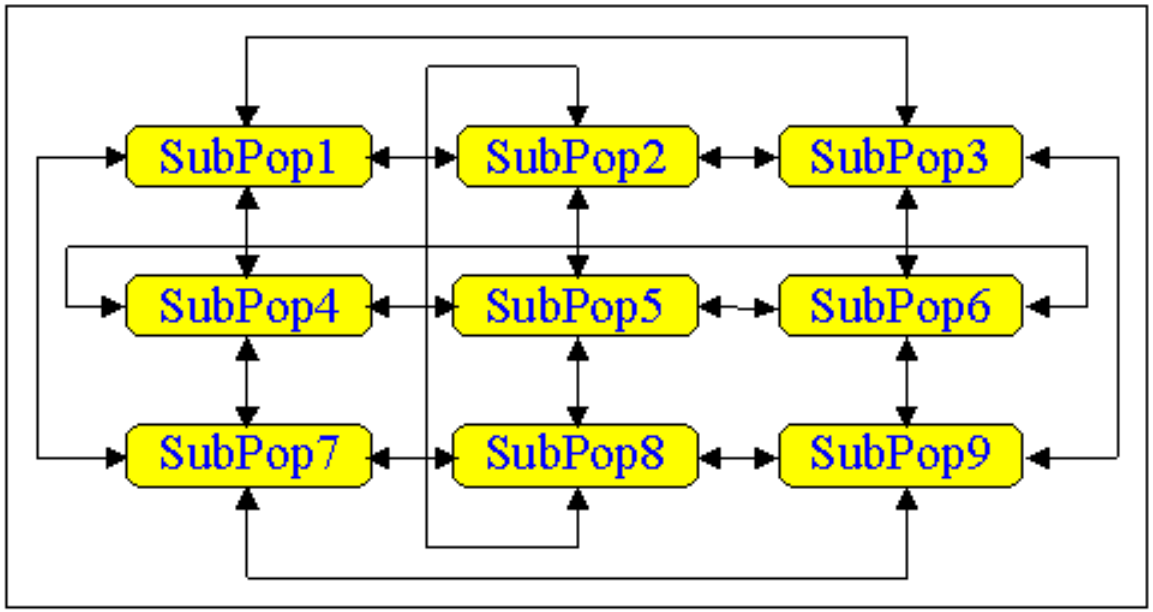


Migrare

Topologie inel

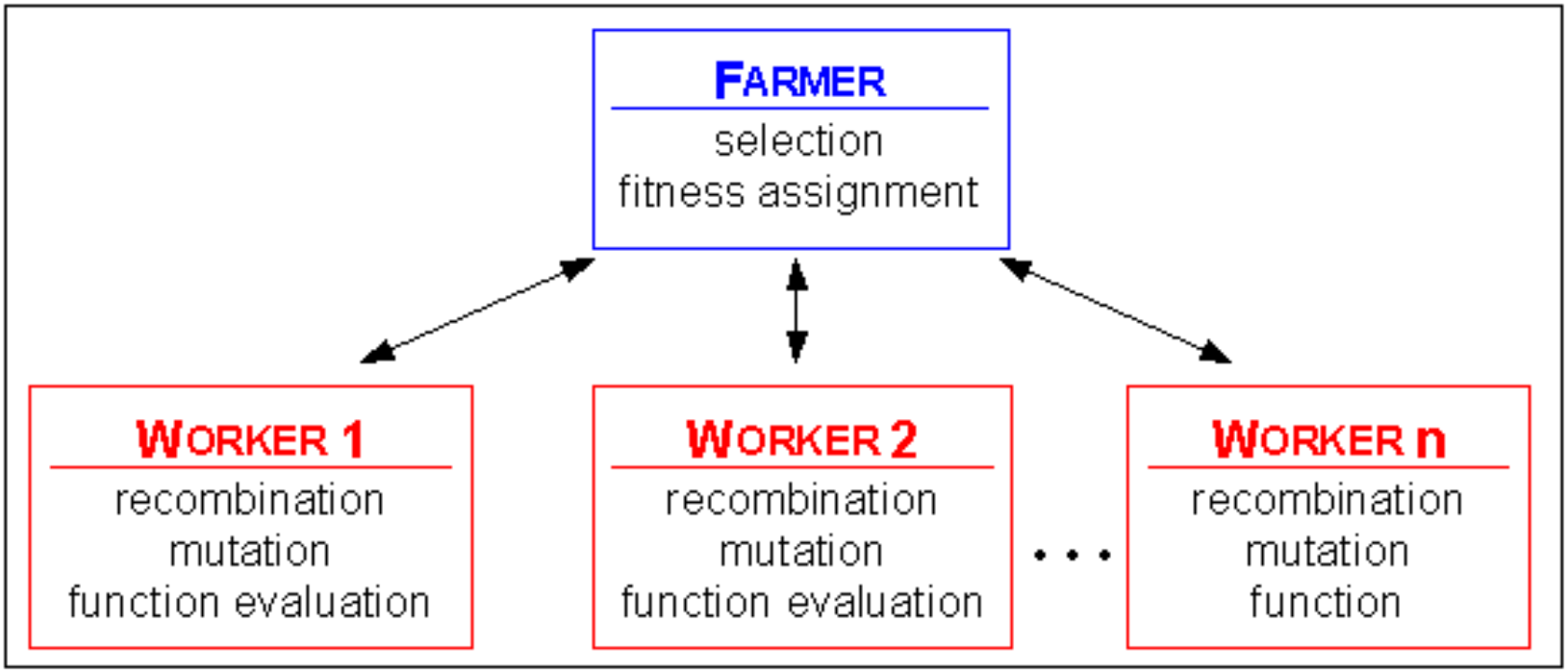


Topologie vecintate (implemantare 2-D)



8.2 Modelul global - worker/farmer

- Exploateaza paralelismul inerent al GA
- Worker/Farmer – schema de migrare



Invatare automata

Universitatea Politehnica Bucuresti
Anul universitar 2021-2022

Adina Magda Florea

Curs Nr. 10

Aplicatii ale algoritmilor genetici

1. TSP cu algoritmi genetici

2. Utilizarea AG in planificare

3. Co-evolutie cooperativa

4. Utilizarea AG in invatarea regulilor de decizie

5. Programare genetica

6. AG pentru generarea de imagini

1. Utilizarea GA pt:

- **Problema 0/1 Knapsack**
- **TSP**

1.1 0/1 Knapsack Problem

- Se da o multime de obiecte, fiecare cu o greutate/pondere **w(i)** si valoare/profit **p(i)**.
- Sa se determine numarul de obiecte din fiecare tip care sa se includa intr-o colectie a.i. greutatea sa fie mai mica decat o valoare data **W** si valoarea totala sa fie maxima.
- Problema 0/1 knapsack - 0 sau 1 obiecte din fiecare tip.
- *Multiobjective optimization problem*: maximizeaza profit si minimizeaza greutate
- Nu exista o (singura) solutie optima ci un set de solutii cu "trade-off" optim = multimea de solutii pt care nu se poate imbunatati un criteriu fara a se inrautati altul

0/1 Knapsack Problem

- Maximizeaza **sum(p(i)*w(i))**
- Restrictie **sum(p(i)*w(i)) <= W**
- x(i) = 0 or 1

Sir binar - lungime = numarul de obiecte.
Fiecare obiect are asociata o pozitie in sirul binar

0 – obiectul nu este in solutie

1 – obiectul este in solutie

Operatori genetici:
- selectie turneu
- one-point crossover
- bit-flip mutation.

Dimeniune populatie 100

Maximizeaza **sum(w(i)*p(i))**

Tour 2

Probabilitate mutatie: 0.01

greutate (w) 3 3 3 3 3 4 4 4 7 7 8 8 9

valoare (p) 4 4 4 **4** 4 5 5 5 **10 10** 11 11 13

Generation 74:

No	Fit	Cromozom
00	24	000**1**0000**11**000
01	23	0110100000100
02	23	0010100101000
03	23	0110010001000
04	23	0110000110000
05	23	0101010001000
06	23	1010100000100
07	23	0110010001000
08	23	0110010010000
09	23	1010101011100
10	22	0000010011000
11	22	1010101100000
12	22	0110101100000
13	22	1010101100000
14	22	1010101100000
15	22	1010101011100
16	15	0000010001000
17	12	0110010011000
18	10	0010100101010
19	-18	0110011110011

1.2 TSP – Reprezentare problema

Functie de evaluare

- Functia de evaluare pentru N orase este suma distantei euclidiene

$$Fitness = \sum_{i=1}^{N} \sqrt{(x_i - x_{i-1})^2 + (y_i - y_{i-1})^2}$$

Reprezentare

- individ = reprezentare a caii, in ordinea de parcurgere a oraselor

3	0	1	4	2	5
0	5	1	4	2	3
5	1	0	3	4	2

TSP Operatori genetici

Crossover

- Nu se potrivesc operatorii traditionali la TSPs

- Inainte de crossover

  1 2 3 | 4 5 0 (parent 1)

  2 0 5 | 3 1 4 (parent 2)

- Dupa crossover

  1 2 3 | 3 1 4 (child 1)

  2 0 5 | 4 5 0 (child 2)

- Greedy Crossover - Grefenstette in 1985

TSP Operatori genetici - recombinare

Parinti **1 2 3 4 5 0** si **4 1 3 2 0 5**

- Se genereaza un descendent utilizand cel de al doilea parinte ca sablon: selectez orasul **4** ca primul oras al copilului 4 x x x x x
- Gasesc legaturi de la oras 4 in ambii parinti: (4, 5) si (4, 1). Daca distanta (4,1) mai mica decat (4,5), selectez **1** ca urmatorul oras din copil: **4 1 x x x x**
- Gasesc legaturi de la oras 1 in ambii parinti: (1, 2) si (1, 3).
- (1,2) < (1,3) – selectez **2** ca oras urmator: **4 1 2 x x x**
- (2, 3) > (2, 0) – selectez **0: 4 1 2 0 x x**
- (0, 1) < (0, 5). Deoarece 1 apare deja in copil, selectez **5 - 4 1 2 0 5 x**
- Legaturi 5 sunt (5, 0) si (5, 4), dar atat 0 cat si 4 apar in copil. Alegem un oras neselectat 3 - copil **4 1 2 0 5 3**

Aceeasi metoda pt a genera celalat descendent **1 2 0 5 4 3**

TSP Operatori genetici

Mutatie

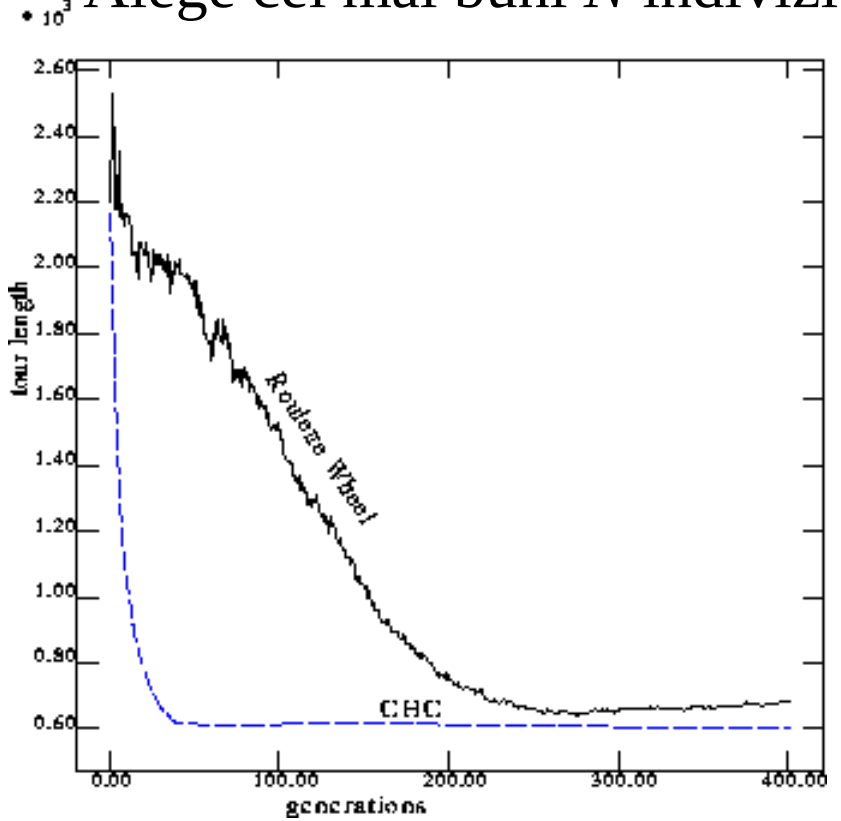
- Nu putem folosi mutatia clasica. De ex: **1 2 3 4 5 0**, mutam 3, schimbam aleator 3 la 5 **- 1 2 5 4 5 0 -** gresit
- Selectam aleator 2 valori si le interschimbam.
- **Swap mutation:** **1 2 3 4 5 0** **1 5 3 4 2 0**

Selectie

- Roulette wheel selection – cel mai bun individ are probabilitatea cea mai mare de selectie dar nu este sigur selectat
- Utilizam **selectia CHC** pt a garanta ca cel mai bun individ supravietuieste (Eshelman 1991).

TSP Comportare

- CHC selection – populatie de dimensiune *N*
- Genereaza *N* copii cu roulette wheel selection
- Combina *N* parinti cu *N* copii
- Ordoneaza 2*N* indivizi in functie de fitness
- Alege cei mai buni *N* indivizi pt generatia urmatoare

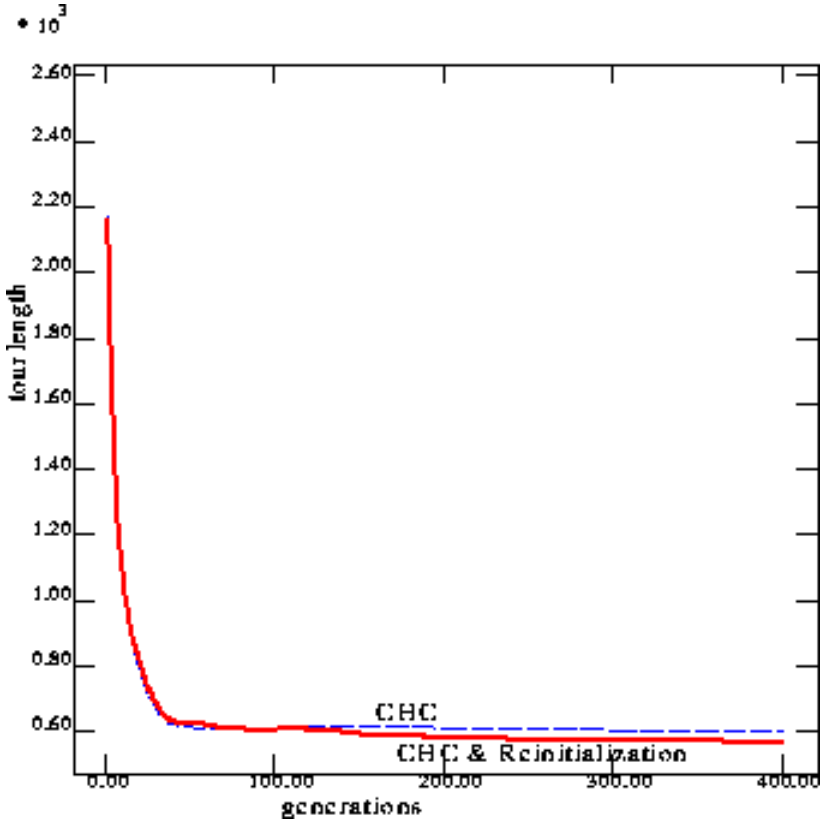


Comparatie
Roulette si CHC selection

Cu selectia CHC populatia converge mai repede decat cu roulette wheel selection si performantele sunt mai bne

TSP Comportare

- Dar convergenta rapida = mai putina explorare
- Pt a preveni convergenta la un minim local, cand am obtinut convergenta populatiei, salvam cei mai buni individi si re-initializam restul populatiei aleator.
- Selectie CHC asfel modificata = selectie **R-CHC**.



Comparatie selectie CHC cu si fara re-initializare

2. Utilizarea AG in planificare

Planificarea resurselor

- Rezolvare cu AG
- Rezolvare cu co-evolutie cooperativa

Problema

- 4 resurse: A, B, C si D
- Fiecare resursa executa un numar diferit de tipuri operatii O1, O2 si O3.
- Pt fiecare resursa se cunoaste:
  - timpul pt executarea unui tip de operatie O_i
  - costul executarii operatiilor
  - capacitatea resursei

Problema

Se dau: 3 job-uri pt: 1400 unitati O1, 1200 unitati O2 si 900 unitati O3

Se cere: planificarea utilizarii resurselor a.i.:

- Cel mai redus cost de productie
- Executarea tuturor unitatilor cerute
- Sa nu se depaseasca capacitatea resurselor

Resursa	**O1** (min)	**O2** (min)	**O3** (min)	**Cost/min**	**Capacitate** (min)
A	2	5	1	30	1000
B	-	4	2	40	4720
C	1	-	-	50	400
D	1	-	2	20	1000

Modelare AG

- **Indivizi** = numarul de unitati din fiecare operatie atribuit fiecarei resurse.

$Nr.O1_A$	$Nr.O2_A$	$Nr.O3_A$	$Nr.O1_B$	$Nr.O2_B$	$Nr.O3_B$	$Nr.O1_C$	…..

- **Functia de evaluare**: functie de cost = costul total al utilizarii resurselor + puncte totale de penalizare pentru depasirea capacitatii resurselor.

Fitness = $Nr.O1_A$*2*30 + $Nr.O2_A$*5*30 + .. +

α*($Nr.O1_A$*2 + $Nr.O2_A$*5 + $Nr.O3_A$*1 – 1000 +…)

trebuie minimizat

- α - ponderea penalizarilor pt violarea restrictiilor
- Solutia se obtine in aprox 60 generatii cu 50 indivizi si o valoare a solutiei de 99% din solutia de cost optim.

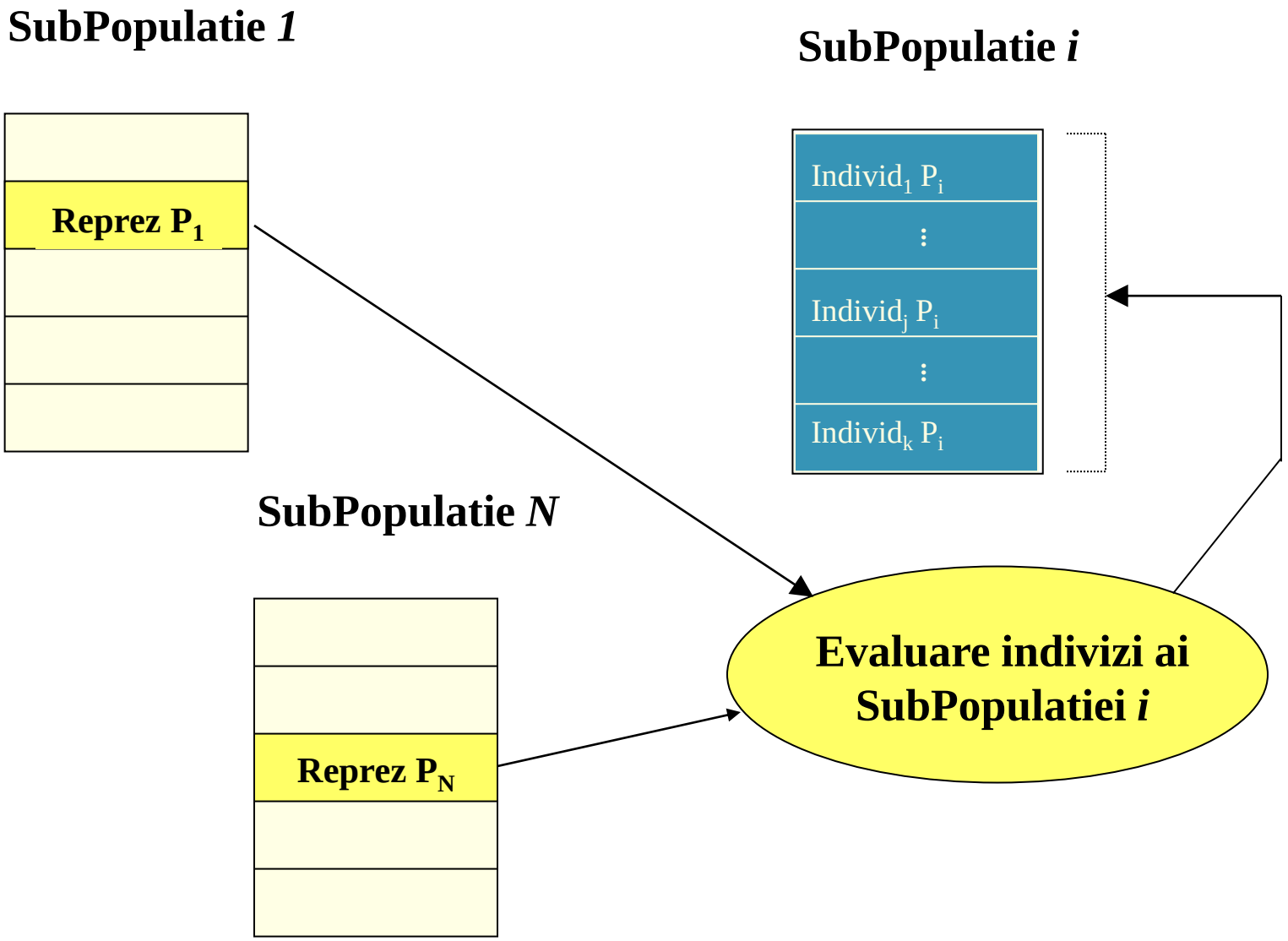
Resursa	O1	O2	O3	Cost/min	Capacitate
A	2	5	1	30	1000
B	-	4	2	40	4720
C	1	-	-	50	400
D	1	-	2	20	1000

3. Co-evolutie cooperativa

- Modeleaza un sistem cu 2 sau mai multe sup-populatii (specii)
- Speciile sunt izolate genetic
- Speciile interactioneaza intre ele si au relatii de cooperare
- Speciile pot fi evoluate secvential sau paralel
- Pt a evalua o specie se formeaza colaborari cu indivizi (reprezentanti) din celelalte specii

3.1 Co-evolutie cooperativa - principii

- Potter, De Jong, 2000
- Alegere reprezentanti
  - cel mai bun (greedy)
  - aleator
- Evolutia genetica a mai multor specii in populatii separate este o solutie la problema mentinerii diversitatii intre componente
- Fiecare populatie incearca sa acopere o "nişă" a solutiei


SubPopulatie 1
Reprez P1
SubPopulatie i
Individ1 Pi
Individj Pi
Individk Pi
SubPopulatie N
Reprez PN
Evaluare indivizi ai SubPopulatiei i

Co-evolutie cooperativa – maximizare functie

- Maximizarea unei functii f(x1,..,xn)
- n sub-populatii, pentru fiecare variabila
- evolueaza independent
- firness – cel mai bun individ din n-1 subpopulatii + individual current – maximizarea functiei

3.2 Problema: acoperirea sirurilor

- **M= Match set** = o multime de *N* vectori binari
- **T = Target set** = o multime de *K* vectori binari, K>>>N
- Fiind dat **T**, sa se gaseasca **M** care "acopera cel mai bine" **T**
- **Puterea de acoperire a unui vector cu P elemente**

$$S(x,y)=\sum_{i=1}^{P}\begin{cases}1 \text{ daca } x_i = y_i\\ 0 \text{ in caz contrar}\end{cases}$$

- **Puterea de acoperire a unei multimi M (match set)**

$$S(M)=\frac{1}{K}\sum_{i=1}^{K}\max(S(m_1,t_i),\ldots,S(m_N,t_i))$$

Problema: acoperirea sirurilor

- **N** Sub-Populatii – N sabloane de siruri
- Fitness individ = calculat ca valoarea Match set ***S(M)*** pentru un set de N vectori in care individul current din subpopulatia ***i*** si cel mai bun individ din restul de ***N-1*** sub-populatii

$$S(M) = \frac{1}{K}\sum_{i=1}^{K} \max(S(\vec{m}_1,\vec{t}_i),\ldots,S(\vec{m}_N,\vec{t}_i))$$

- Reprezentant subpopulatii – cel mai bun din fiecare (initial aleator)

- Dimensiune subpopulatii – 50 indivizi
- two point crossover
- bit flip mutation, prob = 1 /lung individ
- Selectie proportionala

Evaluarea abordarii

- Se definesc niste sabloane de siruri
  - parte fixa (0 sau 1)
  - parte variabila #
- 3 Target sets: T1, T2, T3, fiecare cu 200 siruri a cate 64 biti

T1

```
1111111111111111111111111111111################################
################################1111111111111111111111111111111
```

T2

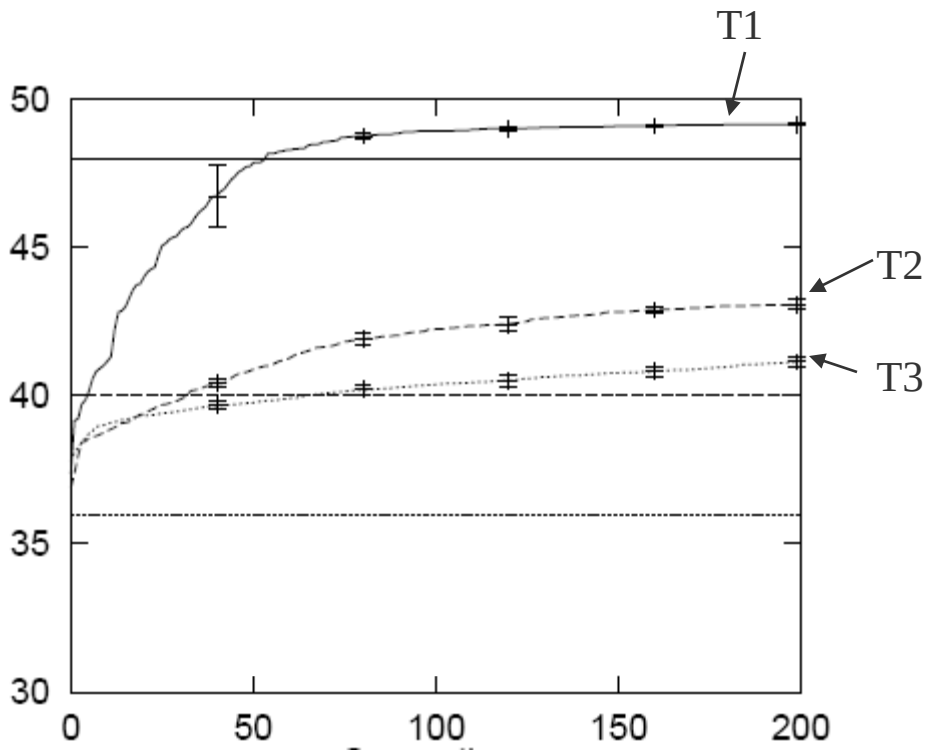
```
1111111111111111############################################################
################1111111111111111############################################
################################1111111111111111##########################
########################################################1111111111111111
```

T3

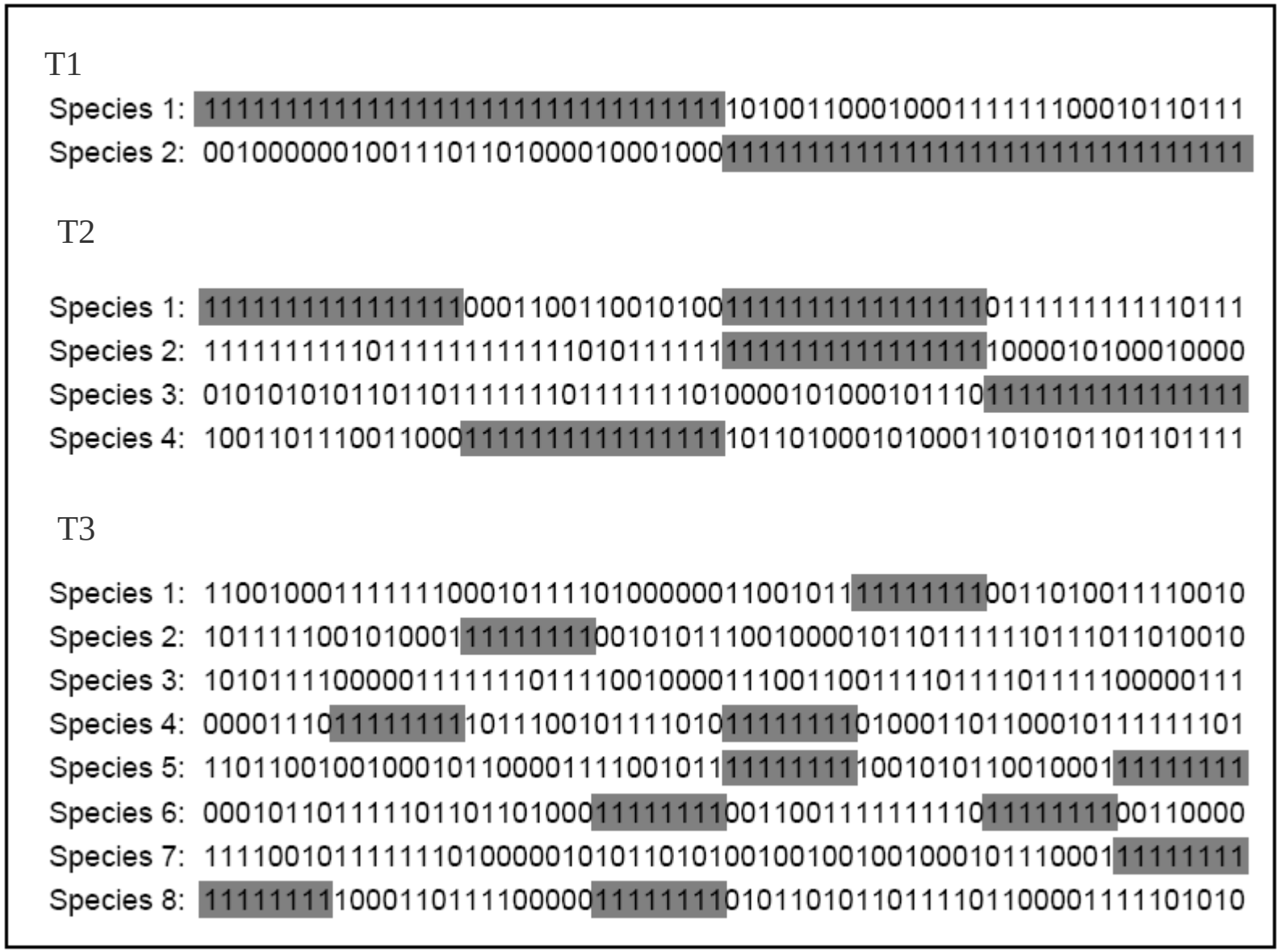
```
1111111#####################################################################
#######1111111#####################################################################
###############1111111#############################################################
############################1111111#################################################
################################1111111###########################################
########################################1111111####################################
########################################################1111111###########
#############################################################1111111#######
#################################################################1111111
```

Evaluarea abordarii

Nr mediu de biti acoperiti



Evaluarea abordarii



Evolutie catre un nivel adecvat de generalitate

- De cate specii este nevoie?
- Exemplu – acoperire optima

```
11111111111111111111111111111111
11111111100000000000000000000000
00000000000000000000001111111111
```

Match set (20+22+22)/3=21.33

```
11111111100000000000001111111111
```

1 Sablon

2 Sabloane

```
11111111111111111111111111111111
10010110110000000000001111110101
```

Match set
(32+20+24)/3=25.33

3 Sabloane

```
11111111111111111111111111111111
11111111100000000000000000000000
00000000000000000000001111111111
```

Evolutie catre un nivel adecvat de generalitate

- Target set – 30 siruri generate din 3 sabloane de 32 biti

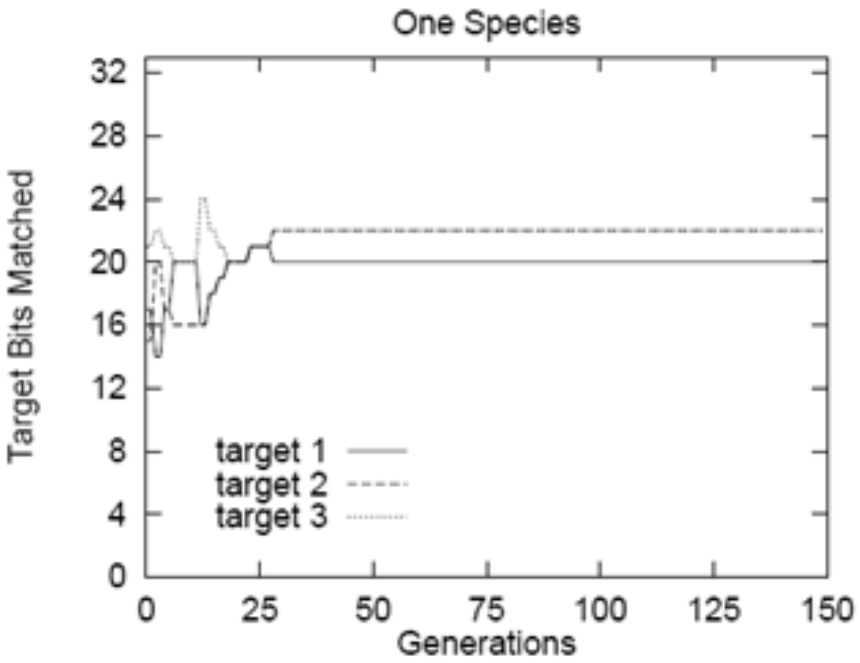
```
11111111111111111111111111111111
11111111100000000000000000000000
00000000000000000000001111111111
```

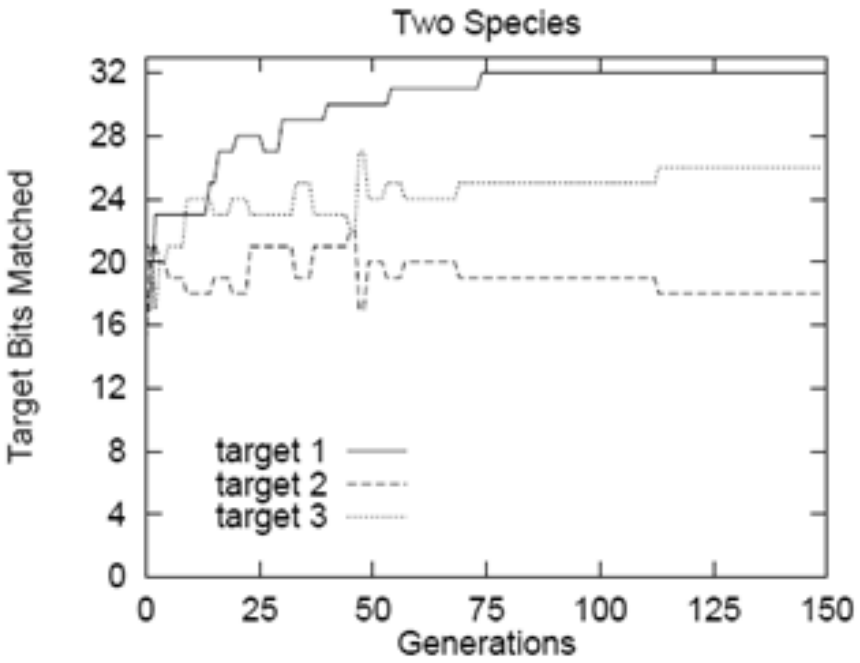
```
1##1###1###11111##1##1111#1##1###1#1111##111111##1#11#1#11######
1##1###1###11111##1##1000#0##0###0#0000##000000##0#00#0#00######
0##0###0###00000##0##0000#0##0###0#0000##001111##1#11#1#11######
```

- Se variaza numarul de specii de la 1 la 4

Evolutie catre un nivel adecvat de generalitate

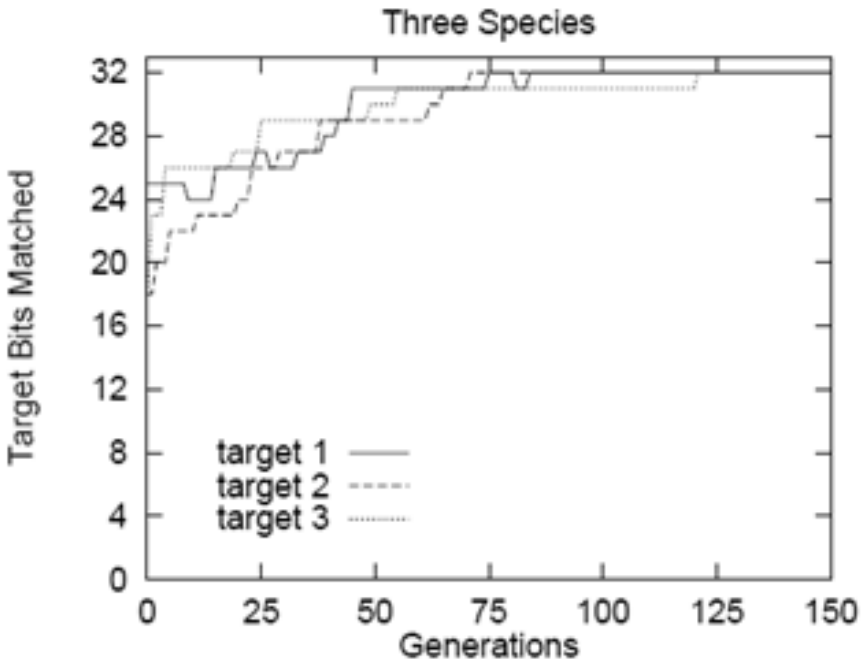
- Potter, De Jong, 2000

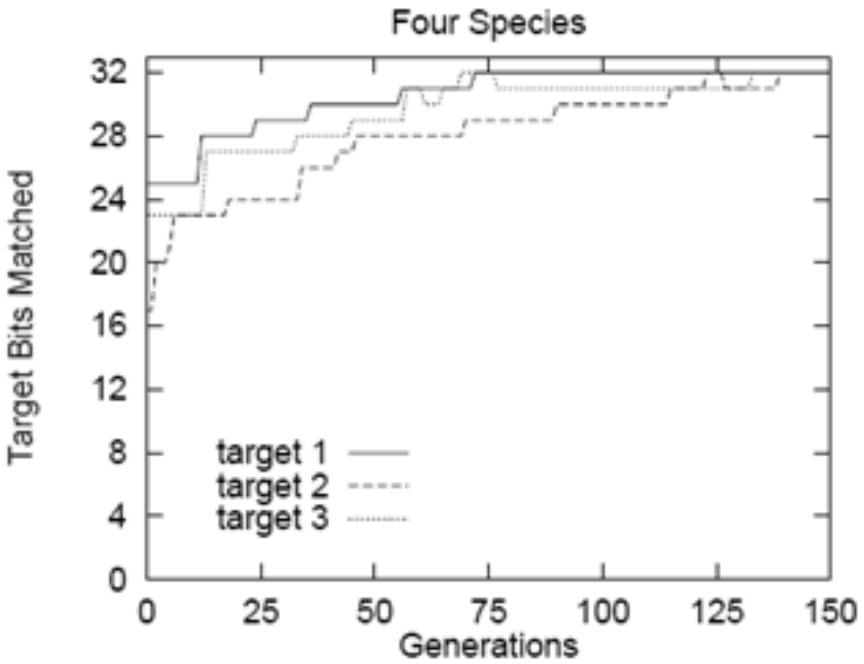




Evolutie catre un nivel adecvat de generalitate

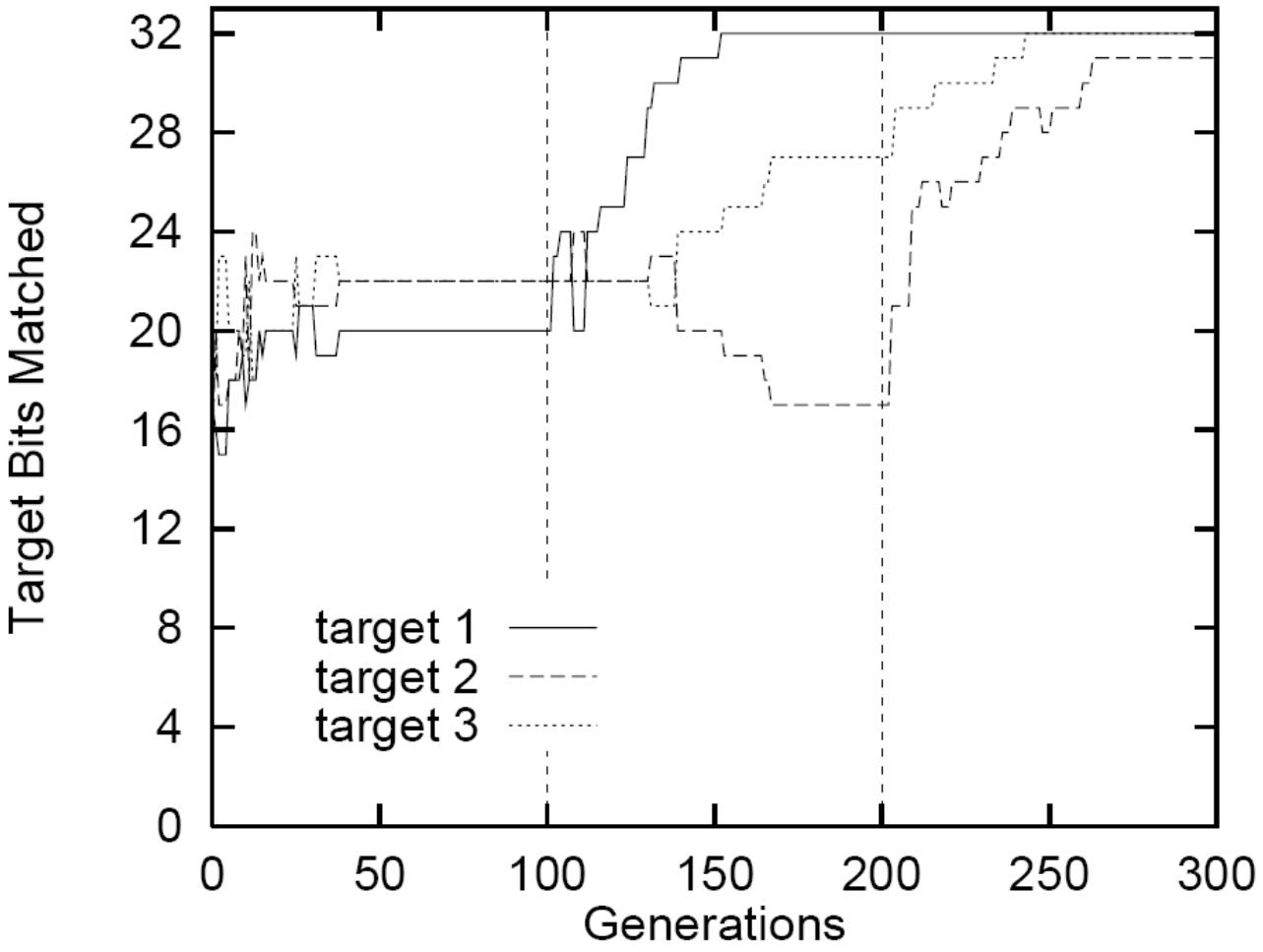
- Potter, De Jong, 2000





Introducere treptata a unor noi subpopulatii

- Potter, De Jong, 2000



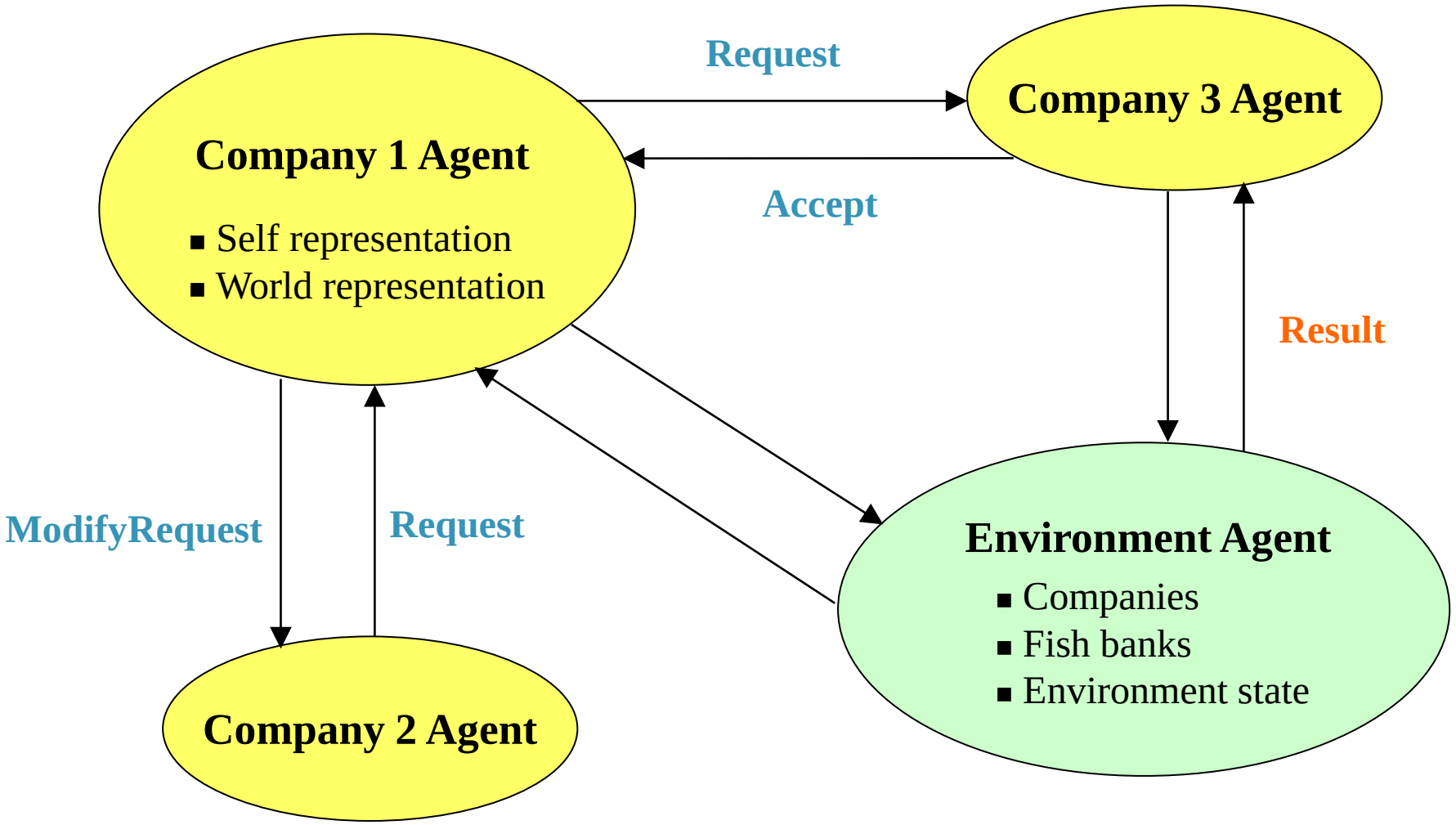
3.3 Problema "tragediei resurselor comune"

- Tragedy of commons
- Exploatare rationala a resurselor regenerabile de catre mai multi agenti
- Motivati individual DAR
- Trebuie sa ajunga la un anumit grad de cooperare

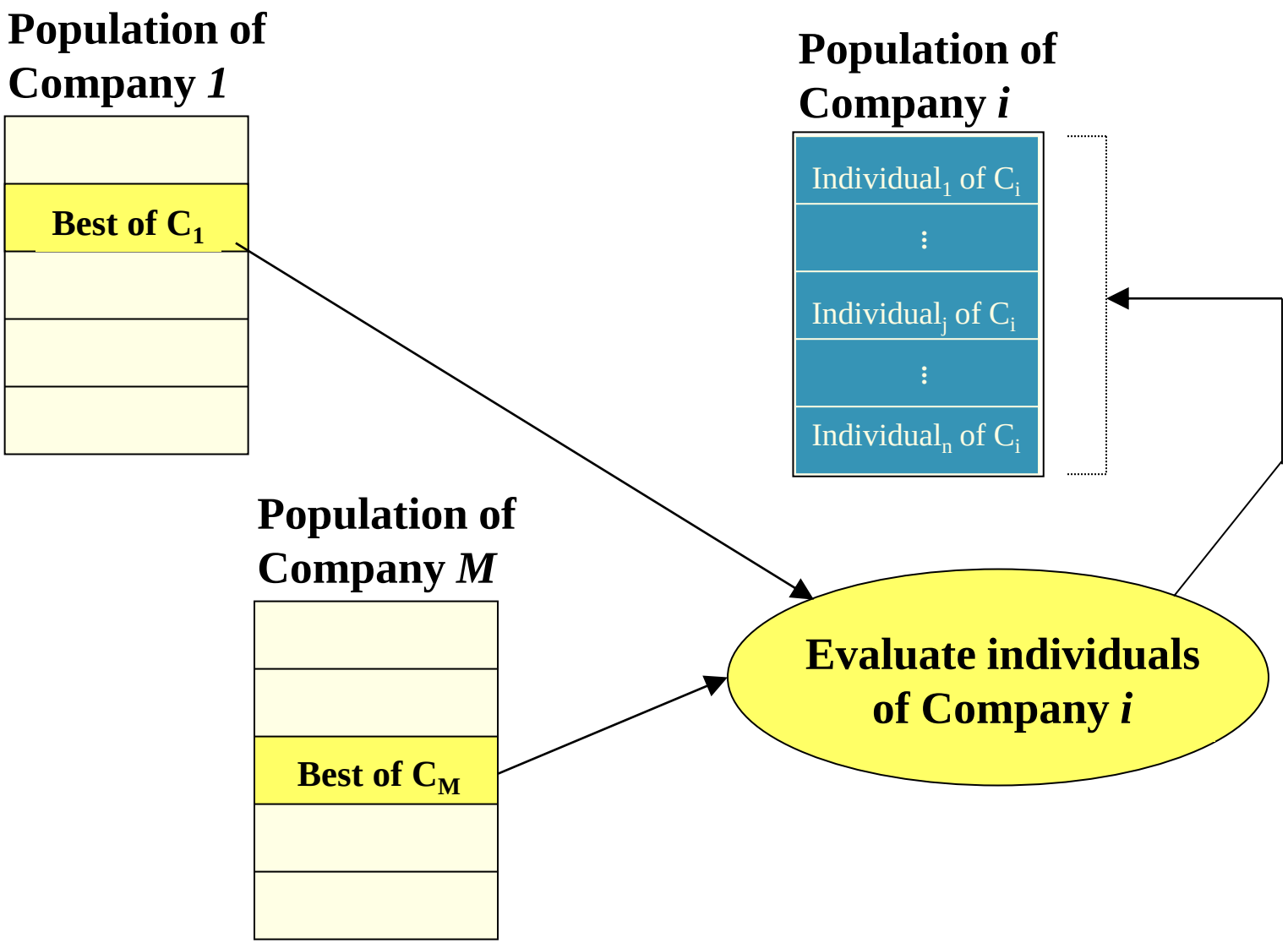
Descriere problema

- O instanta: pescuitul rational
- Companii de pescuit (C_i)
- Fiecare companie are mai multe vapoare de pescuit (NS_i)
- Se poate pescui in mai multe locatii - fishing banks (R_p)
- Se poate pescui in timpul a diferite sezoane (T_j)
- O locatie – resursa regenerabila

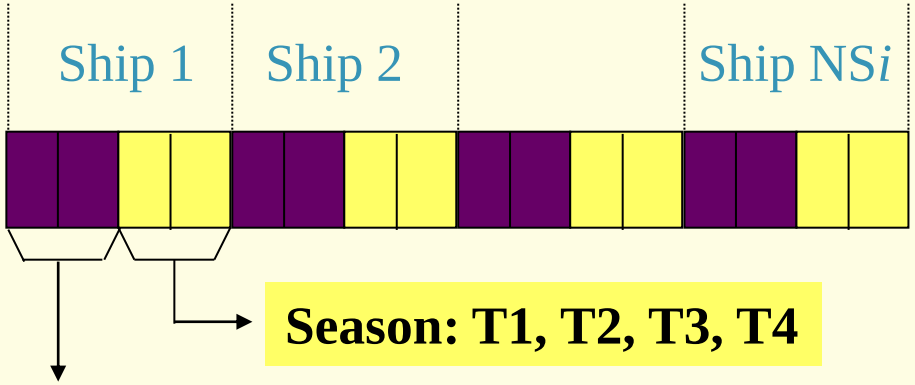
Modelare


Company 1 Agent
Self representation
World representation
Request
Accept
Company 3 Agent
Result
ModifyRequest
Request
Company 2 Agent
Environment Agent
Companies
Fish banks
Environment state

Modelare


Population of Company 1
Best of C_1
Population of Company i
Individual$_1$ of C_i
Individual$_j$ of C_i
Individual$_n$ of C_i
Population of Company M
Best of C_M
Evaluate individuals of Company i

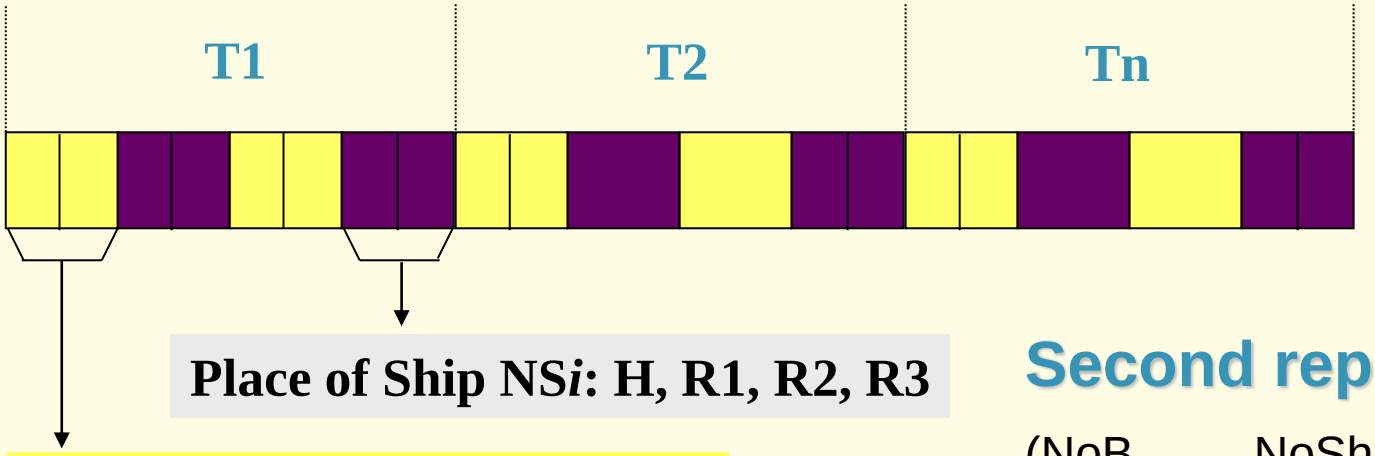
Company *i*



First representation

$(NoB_{Place}, NoB_{Season}) * NoShips$

Company *i*



Second representation

$(NoB_{Place}, NoShips) * NoSeasons$

Parametrii algoritmului genetic:

- Two-point crossover
- Probability of mutation in every individual: **0.1**
- Population size: **100**
- Length of an individual: **100**
- The selection is based on the stochastic universal sampling
- Number of generations: **20..200**

Parametrii algoritmului evolutiv:

- Crossover rate: **0**
- Probability of mutation in every individual: **0.05..0.50**
- Population size: **100**
- Length of an individual: **100**
- Number of generations:**150**

RUN 1					RUN 2				
Company 1	**H**	**R1**	**R2**	**R3**	**Company1**	**H**	**R1**	**R2**	**R3**
Season 1	3	1	5	1	Season 1	2	2	1	5
Season 2	2	3	3	2	Season 2	1	3	3	3
Season 3	1	3	2	4	Season 3	0	3	3	4
Season 4	2	0	3	5	Season 4	3	4	1	2
Season 5	0	3	3	4	Season 5	1	3	0	6
Fitness value	0.836166666666				Fitness value	0.856229166666			
Company 2	**H**	**R1**	**R2**	**R3**	**Company2**	**H**	**R1**	**R2**	**R3**
Season 1	2	4	2	2	Season 1	2	3	1	4
Season 2	1	4	2	3	Season 2	3	3	1	3
Season 3	1	1	2	6	Season 3	4	2	1	3
Season 4	3	2	2	3	Season 4	2	3	4	1
Season 5	0	2	3	5	Season 5	3	2	3	2
Fitness value	0.855166666666				Fitness value	0.730333333333			
Company 3	**H**	**R1**	**R2**	**R3**	**Company 3**	**H**	**R1**	**R2**	**R3**
Season 1	1	4	3	2	Season 1	2	4	1	3
Season 2	3	4	0	3	Season 2	3	2	2	3
Season 3	2	5	3	0	Season 3	1	2	3	4
Season 4	1	1	5	3	Season 4	2	0	5	3
Season 5	2	3	2	3	Season 5	1	0	5	4
Fitness value	0.817166666666				Fitness value	0.826041666666			

Rezultate pentru 3 companii si 2 rulari

4. Utilizarea GA pentru invatarea regulilor de decizie

- Reguli de forma:

  if A_1=v1
  and x1<A_2<=v2
  …
  then Class C_1

- valori de atribute discrete si continue
- Multime de invatare E={e_1,..e_M}
- $e \in E$ este descris prin N atribute A_1, ..A_N si etichetat cu o clasa $C(e) \in C$
- Atribute cu valori discrete – multime finita $V(A_i)$
- Atribute cu valori continue – un interval $V(A_i) = [l_i,u_i]$

4.1 Reguli de decizie

- O clasa $c_k \in C$
- Exemple pozitive - $E^+(c_k) = \{e \in E \mid C(e)=c_k\}$
- Exemple negative - $E^-(c_k) = E - E^+(c_k)$
- O regula de decizie R

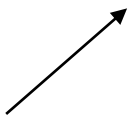
  if t_1 **and** t_2 .. **and** t_r **then** c_k
- LHS
- RHS – apartenenta unui exemplu la o clasa
- Multime de reguli - RS : multimea de reguli care au aceeasi RHS
- $C_{RS} \in C$ – indica clasa specificata de RHS a unei multimi RS
- GA este apelat pt fiecare $c_k \in C$ ca sa gaseasca RS ce separa $E^+(c_k)$ de $E^-(c_k)$
- Functia de fitness -> prefera regulile cu mai putine conditii, care acopera cat mai multe ex+ si cat mai putine ex-

4.2 Reprezentare

- Un cromozon reprezinta un RS (reguli cu acelasi RHS)
- Numarul de reguli din RS nu este cunoscut ➔ Cromozomi de lungime variabila + operatori care sa schimbe numarul de reguli
- 1 cromozom = concatenare de siruri LHS
- Fiecare sir are o dimensiune = LHS a unei reguli de decizie (nu are nevoie de RHS)
- 1 sir este compus din N sub-siruri (LHS) – o conditie pt fiecare atribut

Reprezentare

- Atribute cu valori discrete - flag (0/1 pt fiecare valoare)
- Atribute cu valori continue – l_i, u_i, $l_i<A_i<=u_i$ ($+\infty$, $-\infty$)
- l_i, u_i sunt selectate dintr-o multime finita de valori prag de separare
- Valori prag de separare = punct de separare intre exemple succesive sortate in ordine crescatoare a valorilor A_i - mijlocul intervalului de separare ex+ cu ex-

- ex+ ex+ ex+ | ex- ex- ex- | ex+ ex+ | ex- ex- A_i 

  th_i^{k-1} th_i^{k} th_i^{k+1}
- Daca conditia nu este prezenta – $l_i= -\infty$, $u_i= +\infty$

Reprezentare

Exemplu

- 2 atr valori continue: Salary, Amount
- 1 atr valoare discreta – Purpose (car, house, school)
- Classa - Accept

Salary	Amount	Purpose	
$-\infty$ $+\infty$	$-\infty$ 250	1 1 1	if Amount<250 then ACCEPT
100 250	$-\infty$ 500	1 1 1	if 100<salary<250 and Amount <500 then ACCEPT
750 $+\infty$	$-\infty$ $+\infty$	1 1 0	if Salary>750 and Purpose = (car or house) then ACCEPT

4.3 Operatori genetici

4 operatori aplicati pe un unic set de reguli:

- Schimbarea unei conditii din LHS (a)
- Introducere exemplu pozitiv (b)
- Eliminare exemplu negativ (c)
- Eliminare regula (d)

2 operatori aplicati pe 2 seturi de reguli RS1 si RS2:

- Copiere regula (e)
- Crossover (f)

Operatori genetici

(a) Schimbarea conditiei

- Un operator de tip mutatie – modifica o singura conditie referitoare la un atribut A_i
- Daca A_i discret – selecteaza aleator un flag si inverseaza
- Daca A_i cont – inlocuieste aleator o valoare l_i sau u_i cu o valoare de prag de separare

(b) Introducere ex+

- Modifica o regula R din RS a.i. sa acopere un $e+ \in E^+(C_{RS})$ neacoperit din R
- Toate conditiile din regula care sunt in conflict cu *e+* trebuie sa fie schimbate.
- A_i discret – seteaza flag
- A_i cont – $l_i<A_i<=u_i$ cu $u_i<A_i(ex+)$ – cel mai mic u_i' a.i. $u_i'>=A_i$; similar daca $l_i>= A_i(ex+)$

Operatori genetici

(c) Eliminare ex-

- Modifica o unica regula *R* din setul *RS*.
- Selecteaza aleator un exemplu *e*- din multimea de exemple negative, acoperit de *R*.
- Apoi altereaza o conditie din *R* a.i. *R* sa nu mai acopere *e*-.
- Daca in conditie atr discret *Ai* atunci flag corespunzator *Ai*(*e*-) este pus pe 0.
- Daca *Ai* este atribut cu valori continue *li* < *Ai* <= *ui* este micsorat fie la *li'* < *Ai* <= *ui* fie la *li* < *Ai* <= *ui'*, unde *li'* este cea mai mica valoare de prag de separare *Ai*(*e*-) >= *li'* si *ui'* este cea mai mare valoare de prag de separare a.i. *ui'* < *Ai*(*e*-).

Operatori genetici

Eliminare regula si copiere regula – singurii operatori care schimba numarul de reguli dintr-un set de reguli.

(d) Eliminare regula

- Elimina aleator o singura regula din *RS*.

(e) Copiere regula

- Adauga la *RS*1, o copie a unei reguli selectata aleator din *RS*2, cu conditia ca numarul de reguli din *RS*1 sa fie ma mic decat *maxR*.
- *maxR* este un parametru stabilit de proiectant, care limiteaza numarul maxim de reguli dintr-un set.

(f) Crossover

- Selecteaza aleator 2 reguli R_1 si R_2 din *RS*1 si *RS*2. Apoi aplica crossover intre sirurile ce reprezinta R_1 si R_2.

4.4 Fitness

- Atribuire bazata pe rang
- ***Scop = reducerea numarului de erori***
- E_{RS} – multimea de exemple acoperite de RS (clasificate in clasa C_{RS})
- pos= $|E^+_{RS}| = |E_{RS} \cap E^+(C_{RS})|$ – card multimea de ex+ clasificata corect de RS
- neg = $|E^-_{RS}| = |E_{RS} \cap E^-(C_{RS})|$ – card multimea de ex- acoperita de RS
- Numarul total de ex+ si ex-

  $POS = |E^+(C_{RS})|$

  $NEG = |E^-(C_{RS})|$
- RS clasifica corect daca **POS = pos** si **neg = 0**

  $\mathbf{|E^+(C_{RS})| = |E^+_{RS}|}$ si $\mathbf{|E^-_{RS}|} = 0$

Fitness

F_{error} = Pr(RS) / Compl(RS)

- **Pr(RS)** = probabilitatea de a clasifica corect un exemplu din setul de invatare de multimea de reguli RS
- **Compl(RS)** = complexitatea multimii RS
- **Pr(RS) = (pos + NEG – neg) / (POS+NEG)**
- **Compl(RS) = $(L/N+1)^{\alpha}$**

  L – numarul total de conditii din RS

  N – numarul de atribute

  α - stabilit de proiectant in [0.001..0. 1]
- Trebuie maximizata probabilitate si minimizata complexitatea pt a obtine un set compact de reguli si o clasificare corecta

5 Programare genetica

- Aplicarea algoritmilor genetici asupra unei populatii de programe
- Indivizii = nu sunt gene cu dim fixe ci programe – dimensiune variabila
- Necesita putere mare de calcul
- Sortare, cautare, strategii in jocuri,

Programare genetica

Reprezentarea indivizilor

- Arbori – programare functionala
- Programare genetica liniara – programe in limbaje clasice
- Discipulus – software comercial – utilizeaza cod masina (reprezentare binara)

.

Programare genetica

- Arbori – programare functionala
- Programele = arbori sintactici

  max(x * x, x + 3 * y)
- Noduri interne = functii
- Frunze = terminale
- Programe = compuse din functii
- Set de functii grupate intr-o radacina
- Arborii – notatie prefixata (S-expression)

  (max (* x x) (+ x (* 3 y)))

Pasi pregatitori

Stabilirea reprezentarii si a parametrilor de control

1. Set de terminale (vars, funct cu 0 args, const)
2. Set de functii (aritmetice, conditionale)
3. Fitness (explicit sau implicit)
4. Parametrii de control pentru AG
5. Conditia de terminare si metoda de alegere a rezultatului

Algoritmul genetic

1. Initializare: populatie generata aleator cu programe compuse din functii si terminale
2. Repeta pentru mai multe generatii

- Executa fiecare program si determina fitness
- Selectioneaza cf schemei de selectie
- Creaza noi indivizi prin aplicarea op gen:
  - Reproductie: copiaza individ
  - Crossover: creeaza 2 descendenti din 2 parinti sau 1 parinte
  - Mutatie: asupra unui individ existent in populatie

3. Testeaza conditie de terminare

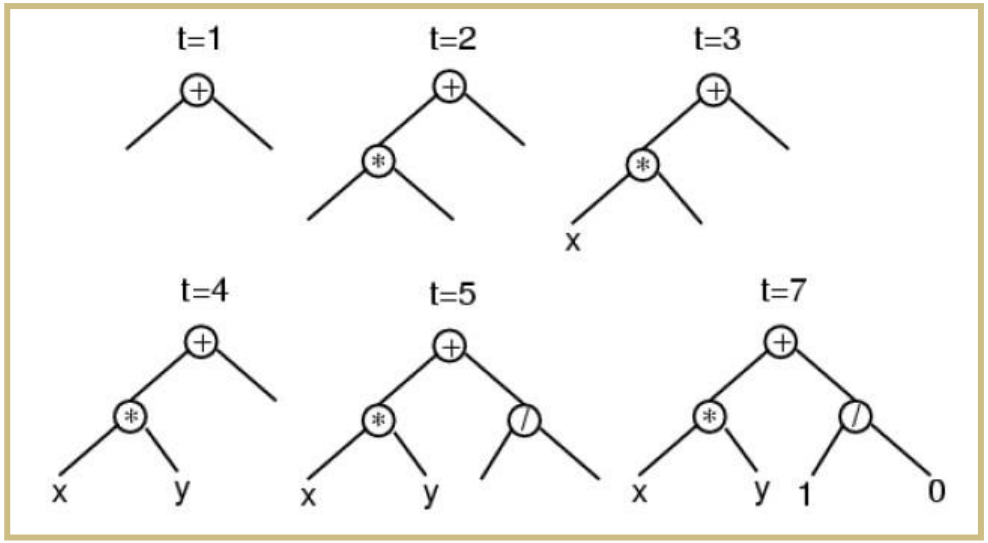
Metoda de initializare

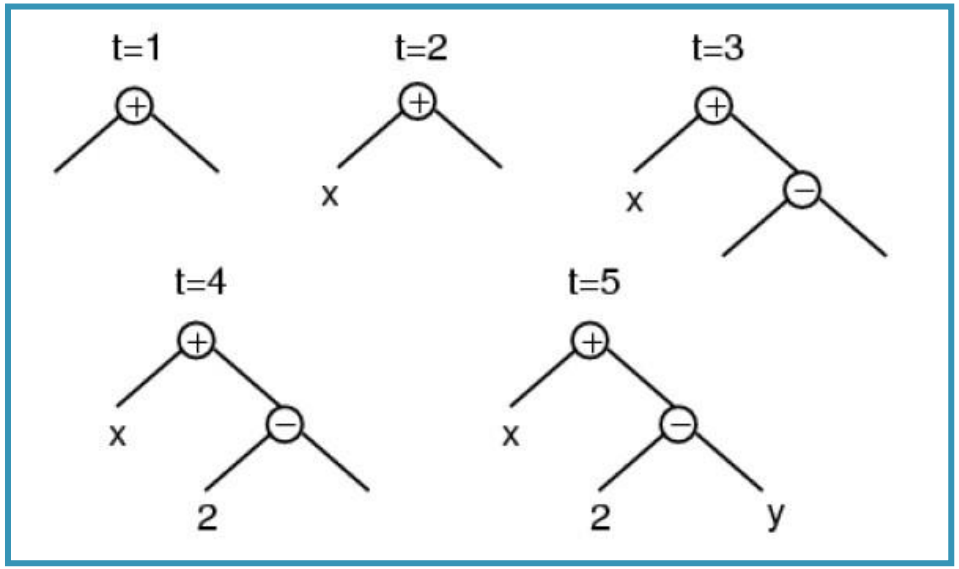
Populatie generata aleator cu programe compuse din functii si terminale

Construieste aleator arborele pana la o adancime maxima *max_d*

1. Metoda “Full” – arbore binar complet cu terminale numai pe ultimul nivel
2. Metoda “Grow” –arbore binar cu terminale pe orice nivel

Full si Grow





Operatori genetici

Crossover 2 parinti = parinti diferiti

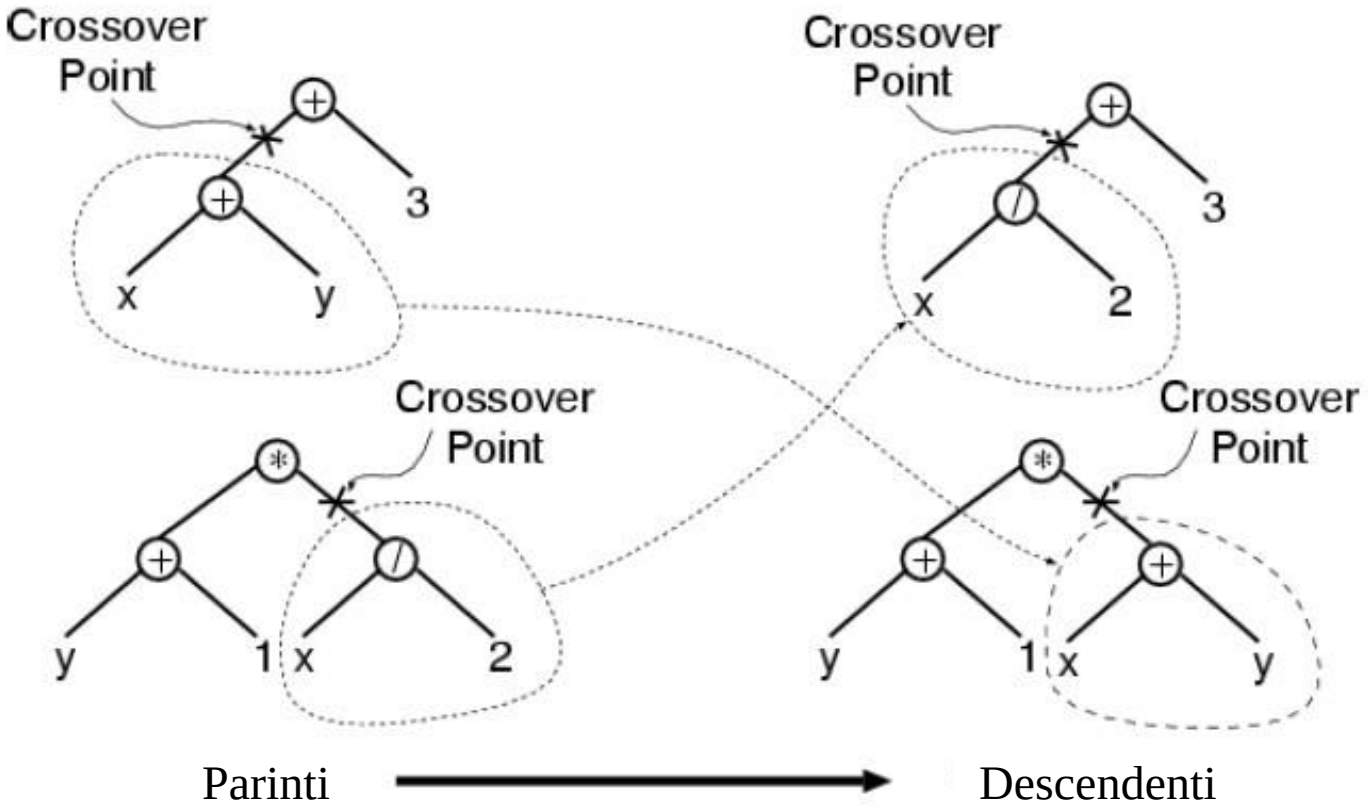
Crossover 1 parinte = parinti identici (reproducere)

Crossover point = 90% nodui interne, 10% frunze

Mutatie = crossover intre un program existent si unul generat aleator

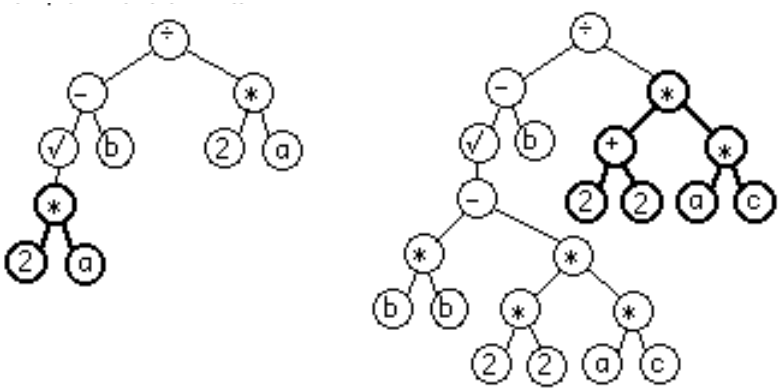
Toate programele obtinute sunt valide din punct de vedere sintactic

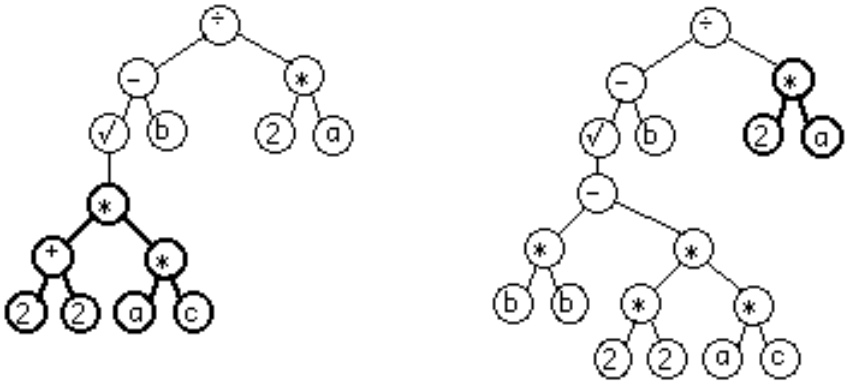
Crossover


Crossover Point
Crossover Point
Crossover Point
Crossover Point
Parinti
Descendenti

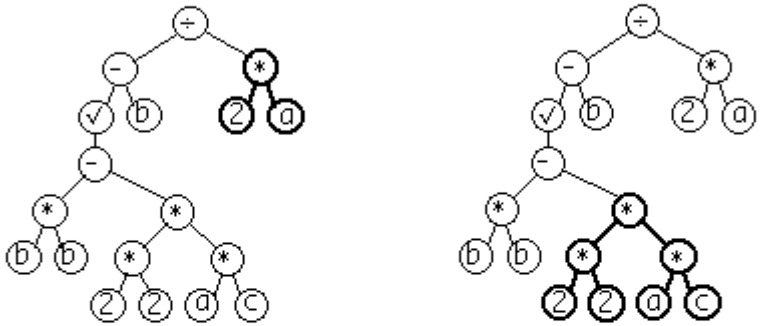
Crossover

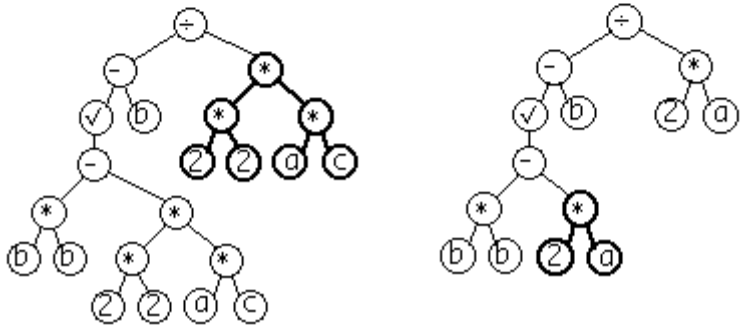
Crossover cu parinti diferiti



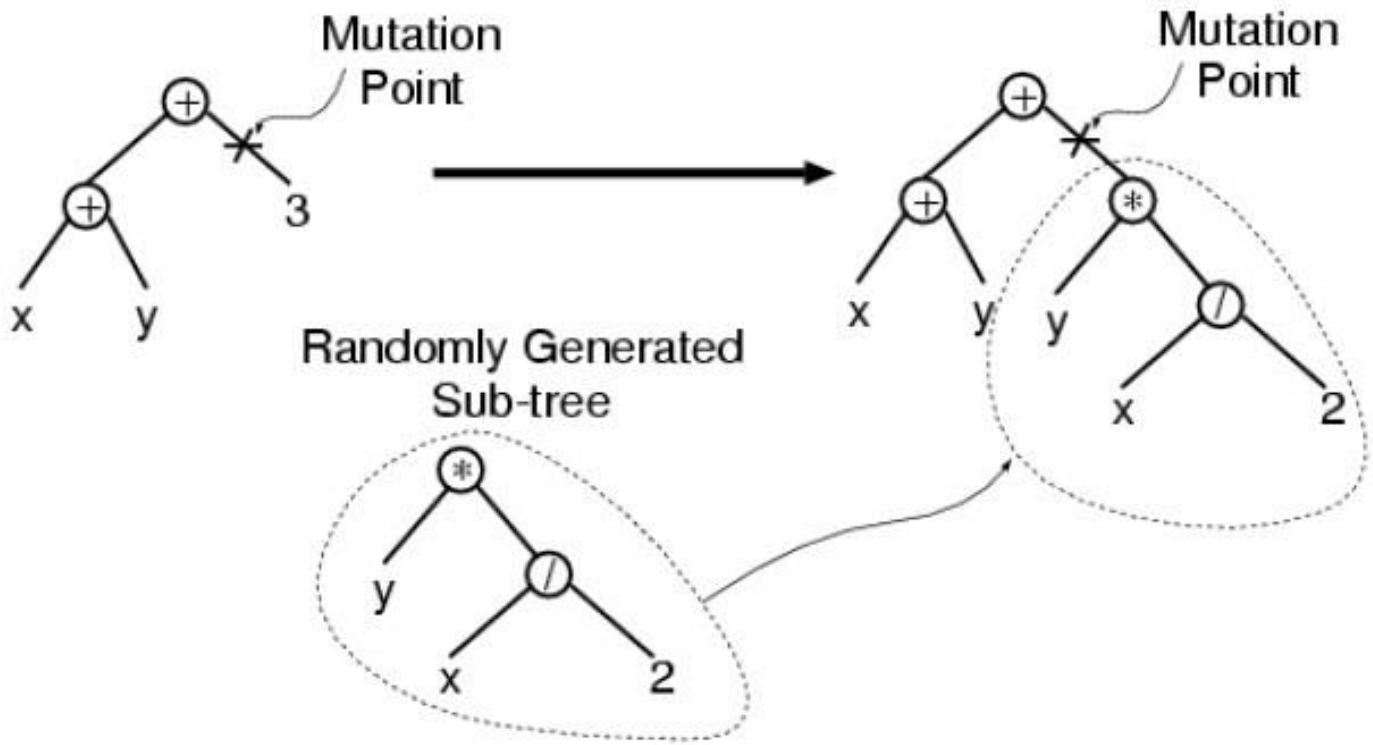


Crossover cu parinti identici





Mutatie


Mutation
Point
3
x
y
Mutation
Point
x
y
y
x
2
Randomly Generated
Sub-tree
y
x
2

Evaluare

Fitness

- eroarea intre iesirea actuala si iesirea dorita
- cat de bine recunoaste anumite sabloane
- castig in jocuri, etc.

Executat pe seturi de valori – fitness cases

Exemplu

Program genetic pt a calcula volarea ecuatiei

$x^2 + x + 1$, cu x in [-1.0, 1.0]

T (terminal set)={x, R}, R = (-5, 5)

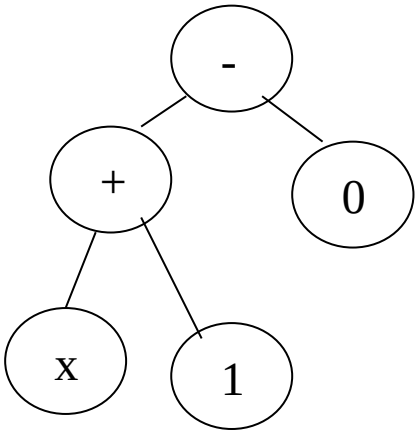
F (function set) = {+, -, *, /} , cu / modif pt 0

Fitness = integrala erorii intre functia rezultata si functia corecta

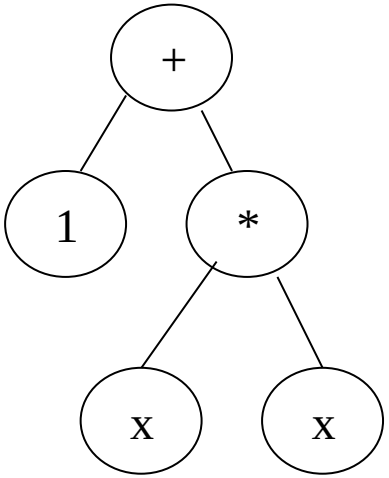
- **Selectie:** turneu, proportionala cu fitness
- **Crossover** – 90 % din populatie
- **Reproductie** - 8%
- **Mutatie** - 2%
- **Fitness** < 0.01
- Metoda de initializare Grow

Exemplu

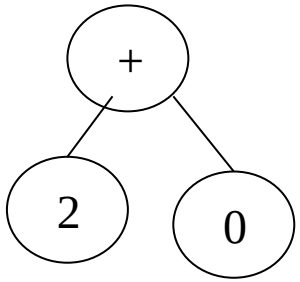
(a) **x+1**



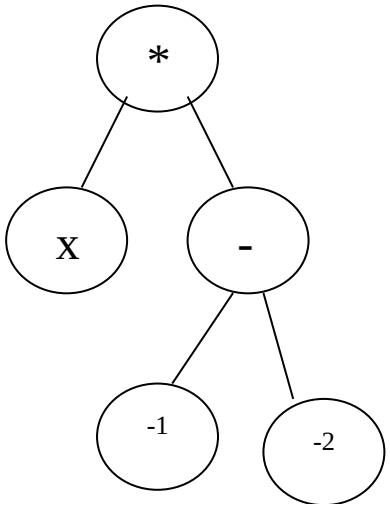
(b) **x^2+1**

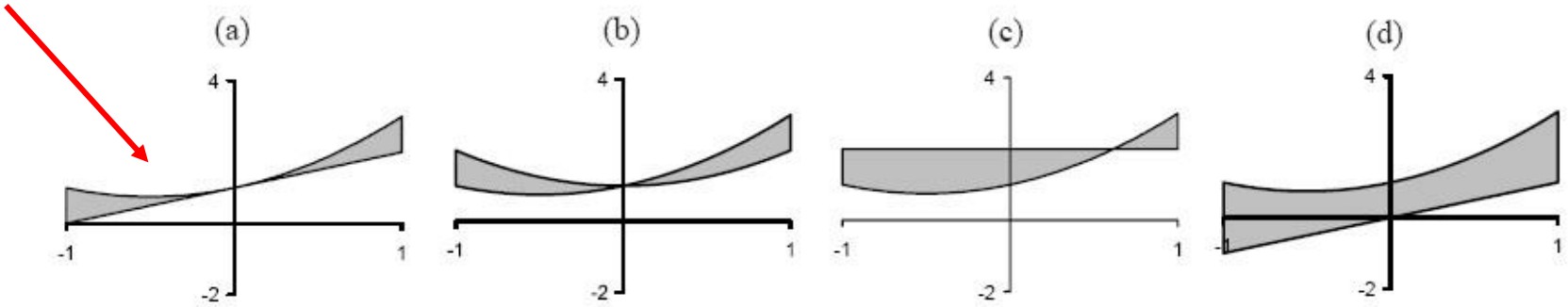


(c) **2**



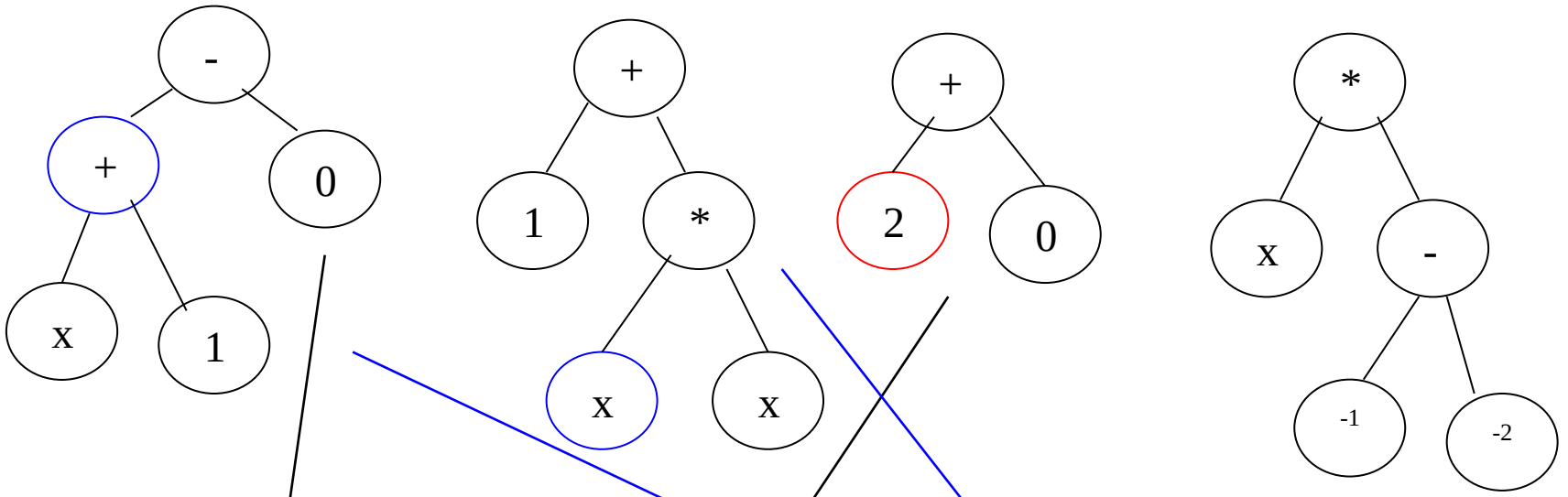
(d) **x**



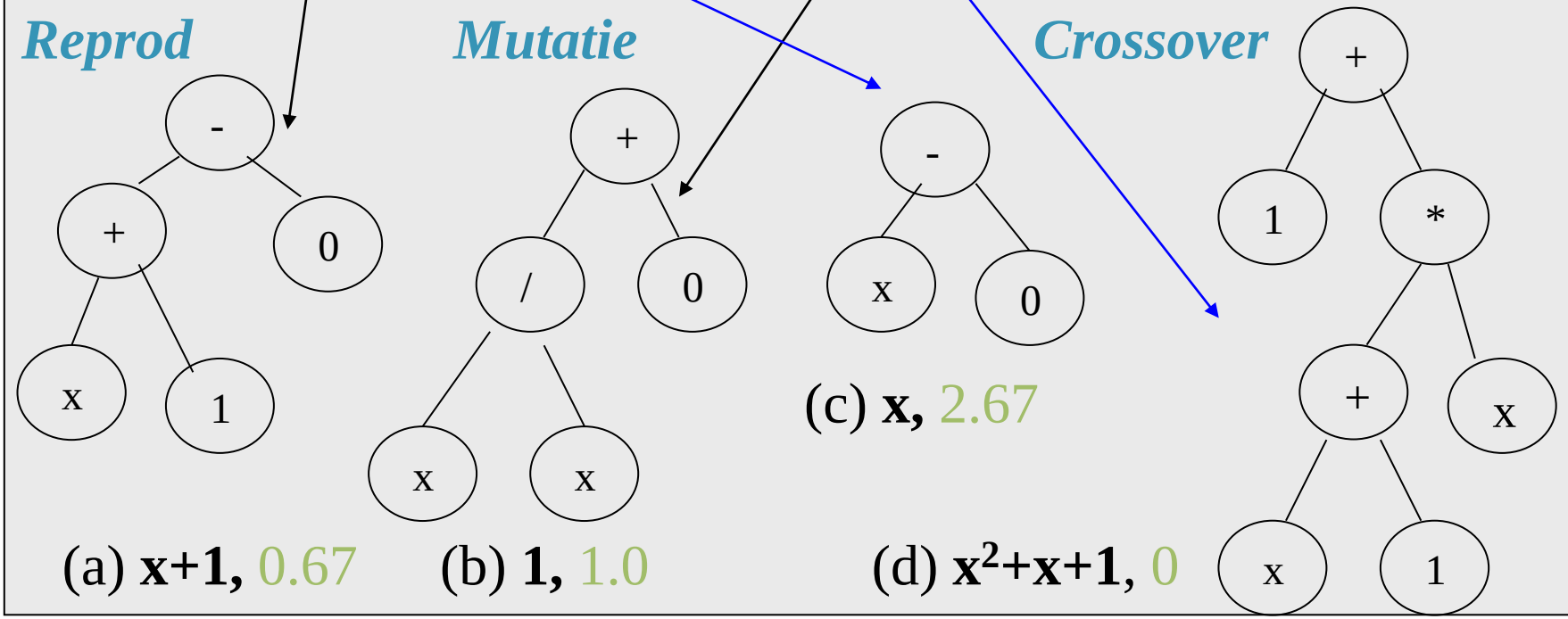


0.67 1.0 1.67 2.67

(a) **x+1,** 0.67 (b) **x²+1,** 1.0 (c) **2,** 1.67 (d) **x,** 2.67



Reprod *Mutatie* *Crossover*

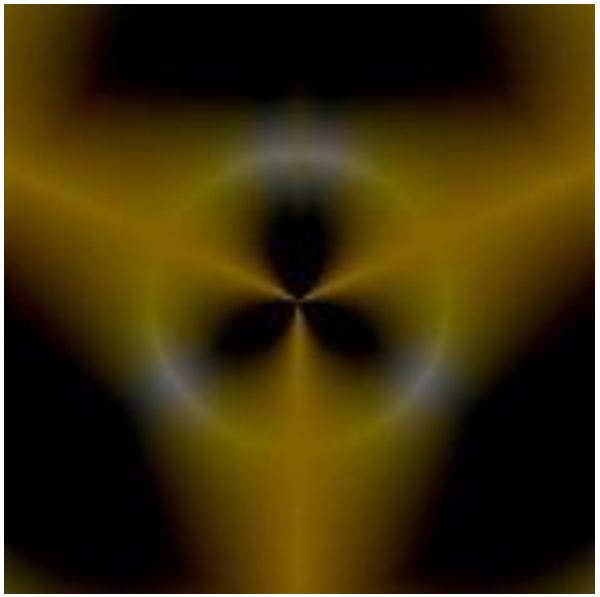


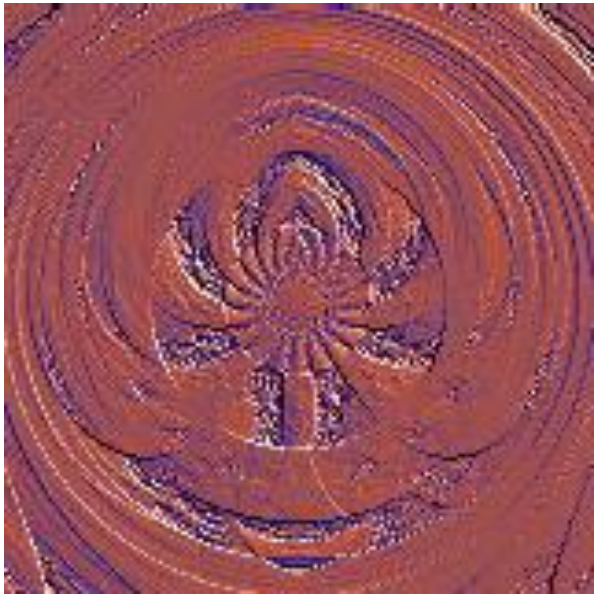
(a) **x+1,** 0.67 (b) **1,** 1.0 (c) **x,** 2.67 (d) **x²+x+1,** 0

6 AG pentru generare de imagini

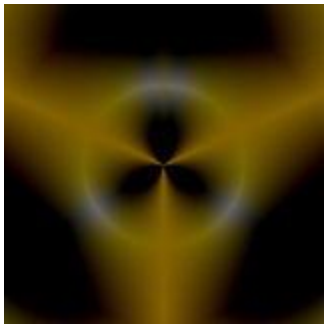
- Programul *Gaia* – generarea de noi imagini
- Fiecare imagine este generata prin evaluarea unei formule matematice
- Sa se gaseasca formule care prin evaluare sa produca imagini interesante.
- GA pt a produce astfel de formule
- Se genereaza aleator o formula si se genereaza variatii ale acesteia
- Utilizatorul selecteaza cele mai interesante imagini
- Formulele selectate devin noi generatii; procesul se repeta
- Conceptul de evolutie este utilizat pentru a genera noi formule din cele existente

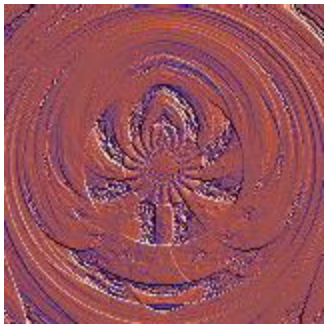
GA pentru generare de imagini





GA pentru generare de imagini

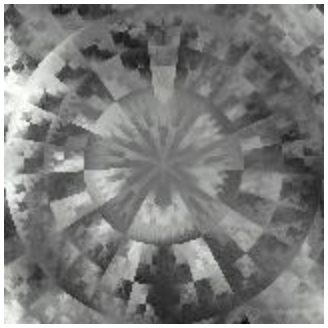




- (lerp #(0.524 0.389 -0.394) (- (triwave (RAD)) 0.590) (PHY))
- (color_grad "earth" (gradient (log (+ (&& (lerp (PHY) #(-0.080 0.408 0.254) (RAD)) (RAD)) #(-0.098 -0.277 -0.840)))))
- (color_noise (mod (warped_color_noise (X) -0.003 -0.296 #(-0.359 0.020 -0.790)) (Y)) (color_noise (X) (invert (Y))))

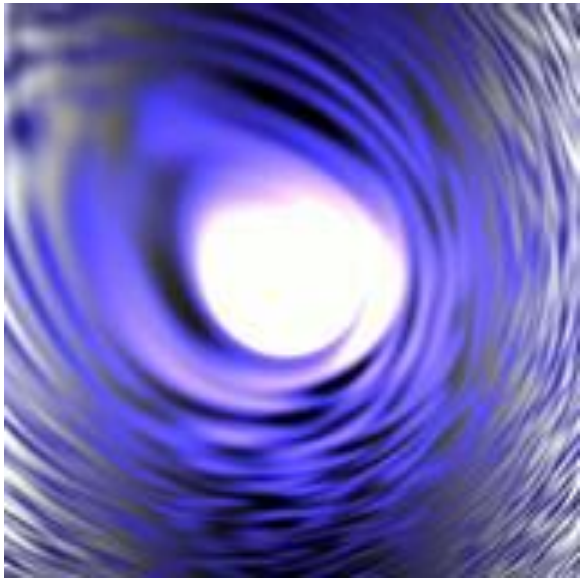
GA pentru generare de imagini

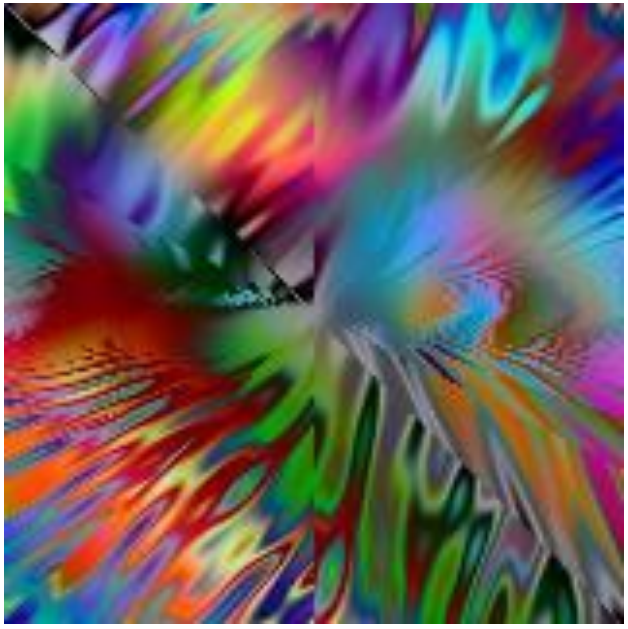
GA pentru generare de imagini



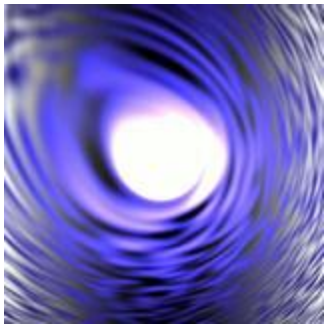
- (lerp (* (vector -0.422 (warped_bw_noise (RAD) #(-0.468 -0.375 -0.624) (Y) (RAD)) (RAD)) (X)) (RAD) (bw_noise (PHY) (log 0.529)))
- (triwave (- (^ (& (RAD) (PHY)) (PHY)) #(-0.176 0.738 -0.928)))
- (mynoise (triwave (|| (RAD) (PHY))) (RAD))

GA pentru generare de imagini



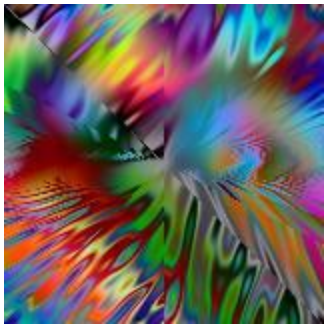


GA pentru generare de imagini



- (lerp (* (X) (X)) (/ (/ #(0.204 0.166 0.711) (RAD)) (RAD)) (bw_noise (RAD) (RAD)))

- (triwave (lerp (min (lerp (PHY) (lerp (PHY) 0.033 (RAD)) #(0.050 0.137 -0.586)) (PHY)) (IRAD) (RAD)))



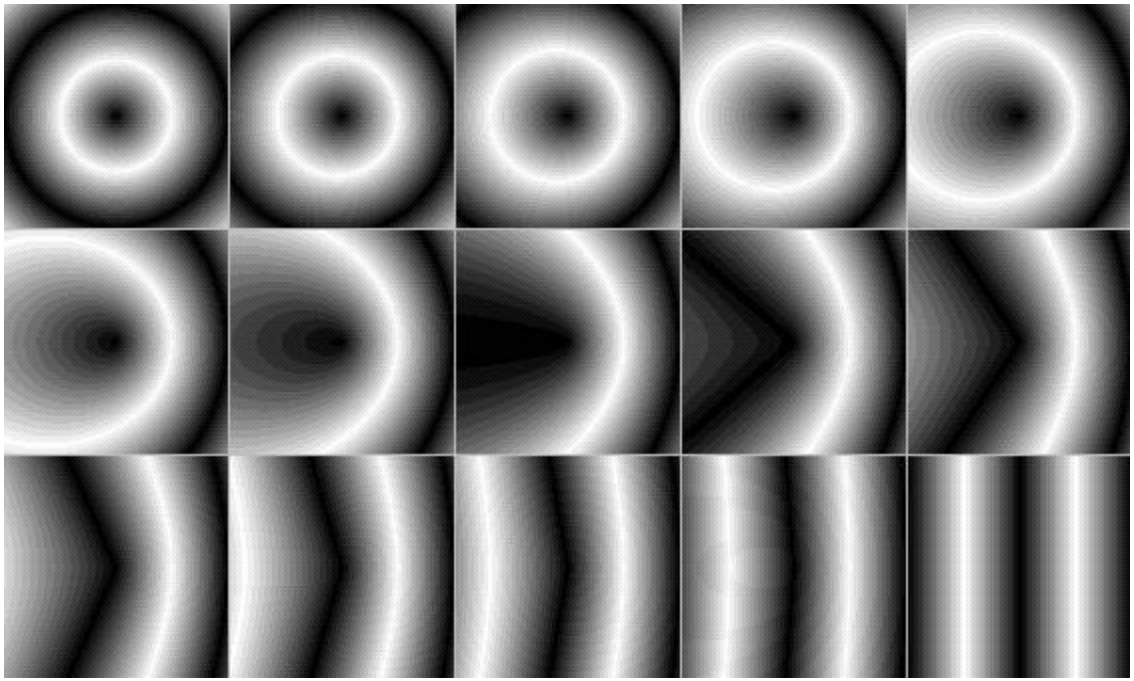
- (color_noise (cos (+ (bw_noise (mod (X) (Y)) #(-0.419 -0.415 0.673)) (Y))) 0.296)

GA pentru tranzitii de imagini

- Programul poate genera de asemenea tranzitii intre imagini generate
- Utilizatorul selecteaza sursa si desinatia si programul gaseste secventa de imagini care face tranzitia
- Daca formulele nu au nimic in comun, tranzitiile sunt pure amestecuri ale imaginilor
- Daca formulele au similaritati se obtin tranzitii interesante

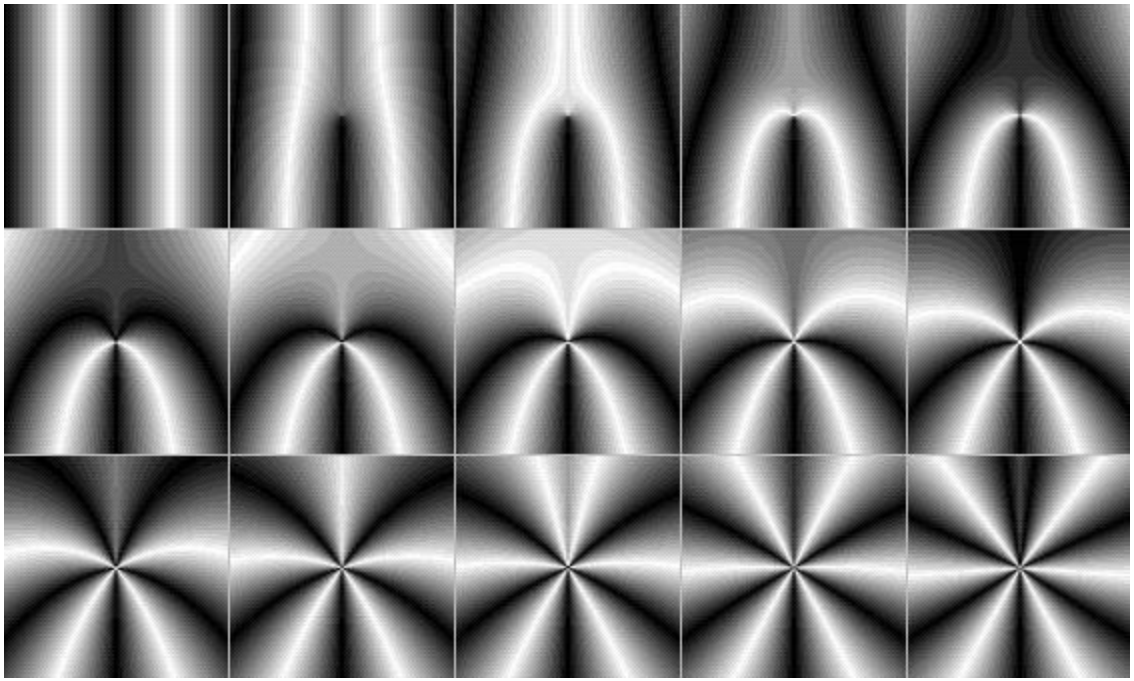
GA pentru tranzitii de imagini

Sursa: (triwave (abs (RAD))) **Dest:** (triwave (abs (X)))



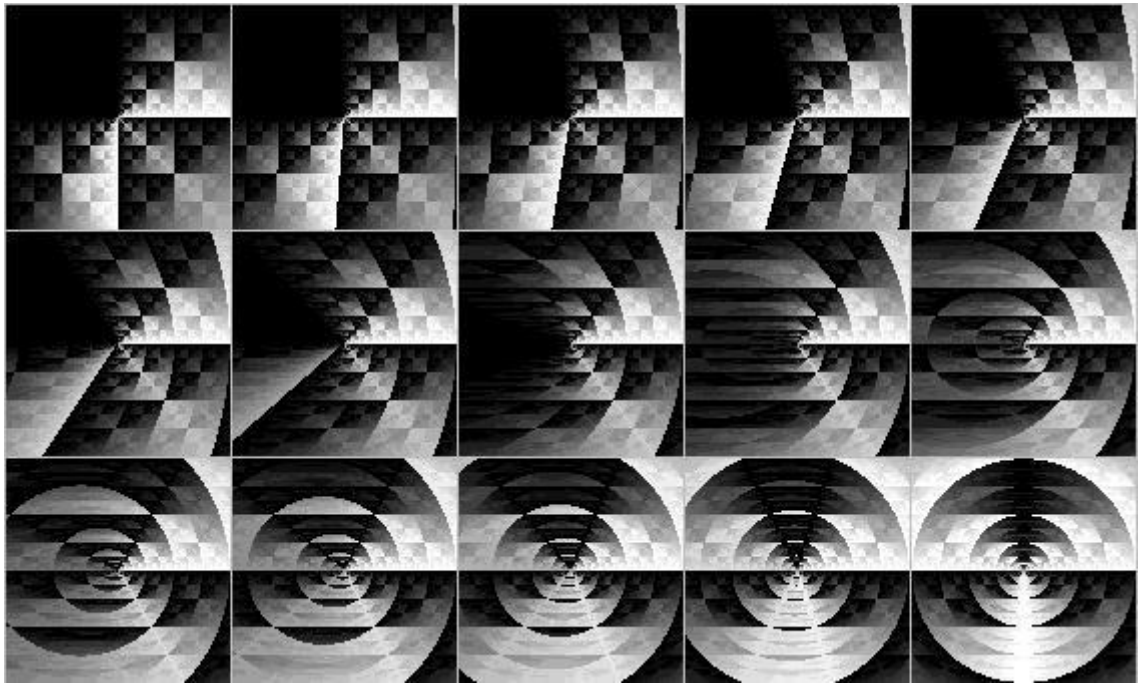
GA pentru tranzitii de imagini

Sursa: (triwave (abs (X))) **Dest:** (triwave (&& (PHY) (PHY)))



GA pentru tranzitii de imagini

Sursa: (/ (& (Y) (X)) (RAD)) **Dest:** (/ (& (Y) (RAD)) (RAD))



Genotip

- Gaia codifica o imagine printr-o formula si un domeniu
- Ambele elemente sunt genotipul solutiei
- Formulele = set de operatori, variabile si constante
- Formula – S-expression
- Domeniul indica unde se evalueaza formula
- Domeniul este o regiune in planul real specificata de limite
- Domeniul implicit este [-1..1] x [-1..1].

Genotip

- Formulele pot fi oricat de lungi
- Operatorii cu argumente – noduri interne
- Operatorii fara argumente – frunze

S-Expression	**Domeniu**
(+ (+ (X) (Y)) (Y))	[0..1] x [0..1]
(gradient (invert (- (0.5) (RAD))))	[-1..1] x [-1..1]
(abs (lerp (noise (/ (triwave (mod (PHY) -0.190)) (RAD)) (min (RAD) -0.873)) (Y) (X)))	[-1..1] x [-1..1]

Formule si operatori

- 5 clase de operatori:

1. ***Operatori de domeniu*: X, Y, RAD, PHY –** depind de domeniul pe care se evalueaza formula.

X, Y

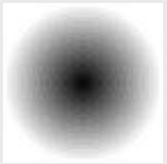
Intoarce o imagine care variaza luminozitatea pixelilor din domeniu fata de axa X sau Y.

Imagini rezultate in domeniul [0..1] x [0..1]

Formule si operatori

RAD

Luminozitatea fiecarui pixel a imaginii este proportionala cu distanta fiecarui pixel din domeniu fata de centru



IRAD

Similar cu RAD dar luminozitatea unui pixel depinde de distanta la cel mai apropiat intreg impar din domeniu

Imagine avand colturile (-1,-1) (-1,1) (1,1) (1,-1)

Formule si operatori

PHY

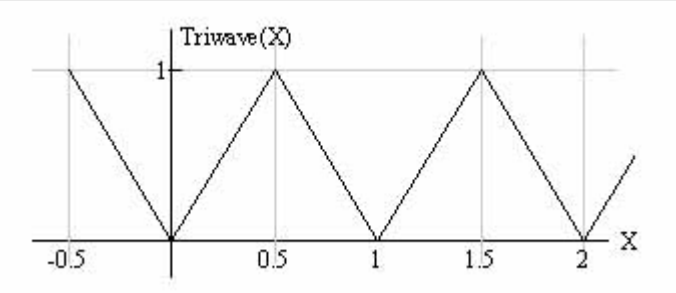
The luminance of the resulting image represents the angular coordinate of the pixel, with the zero heading down.There where the value is greater than 1 a white pixel is used.

Formule si operatori

2. Functii de 1 argument: **cos, sin**, **normalize, gradient, abs, round, triwave, ...**

TRIWAVE

Transforma orice imagine intr-o imagine cu pixeli in intervalul [0,1]



Formule si operatori

3. *Operatori de combinare simpla*: combina 2 imagini si intorc combinatia

Adunare, scadere, multiplicare sau operatii logice

4. Operatori de combinare complexa: au mai multe imagini ca argument; una din imagini determina parametrii combinarii

LERP, noise functions, color grad, ...

5. Operatori diversi: Imagini importate sau sabloane de imagini

Formule si operatori

LERP

Trei imagini A,B,C si calculeaza imaginea rezultata ca:

A+(B-A)*Triwave(C).

Realizeaza o interpolare liniara de la A la B folosind C ca pondere ainterpolarii

De obicei se include Triwave pentru a limita pe C in [0..1]

Mutatie

Evolutia se realizeaza prin 2 operatori:

- Mutatie
- Combinare

Mutatie

- Mutatia se aplica pe un singur nod ales aleator din arbore (eventual si pe noduri descendente ale acestuia).
- Imaginea rezultata area spect apropiat cu ce dinaintea mutatiei

Operatori de mutatie

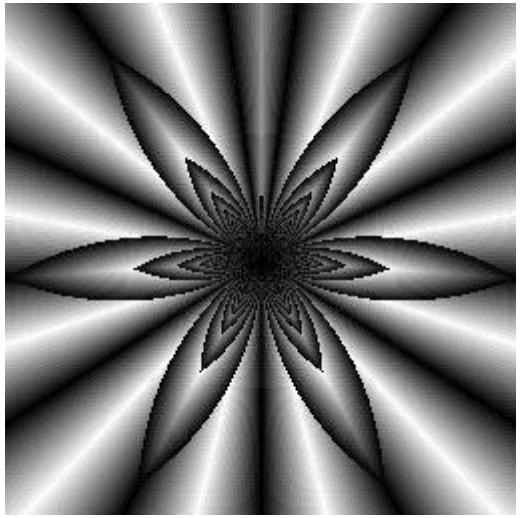
- ***Nod nou:*** Nod selectat inlocuit cu nod generat aleator. Schimbari importante daca nod aproape de radacina
- ***Ajustare nod:*** Schimbare oprator din nod sau schimbare valoare constanta – schimbari miciu
- ***Nod ca argument*:** Un nod devine argumentul unui nod ales aleator, acesta din urma inlocuieste nodul selectat; se genereaza eventual noi operatori
- ***Argument ca nod*:** Un argument dintr-un nod inlocuieste nodul selectat
- ***Nod ca unchi:*** un nod este inlocuit cu unul superior in arbore
- ***Reordonare argumente:*** schimba ordinea argumentelor unui nod selectata (daca are 2 argumente si daca schimbarea este legala)

Exemple de mutatie

- Parinte:

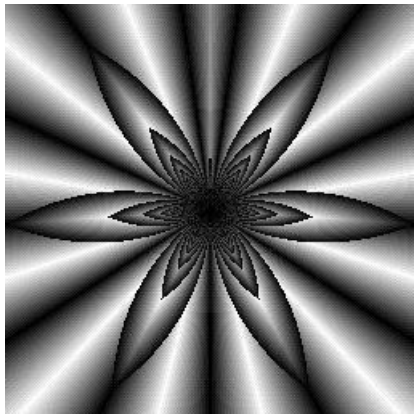
(triwave (mod (triwave (PHY)) (RAD)))

- Domenie: [-1..1] x [-1..1]

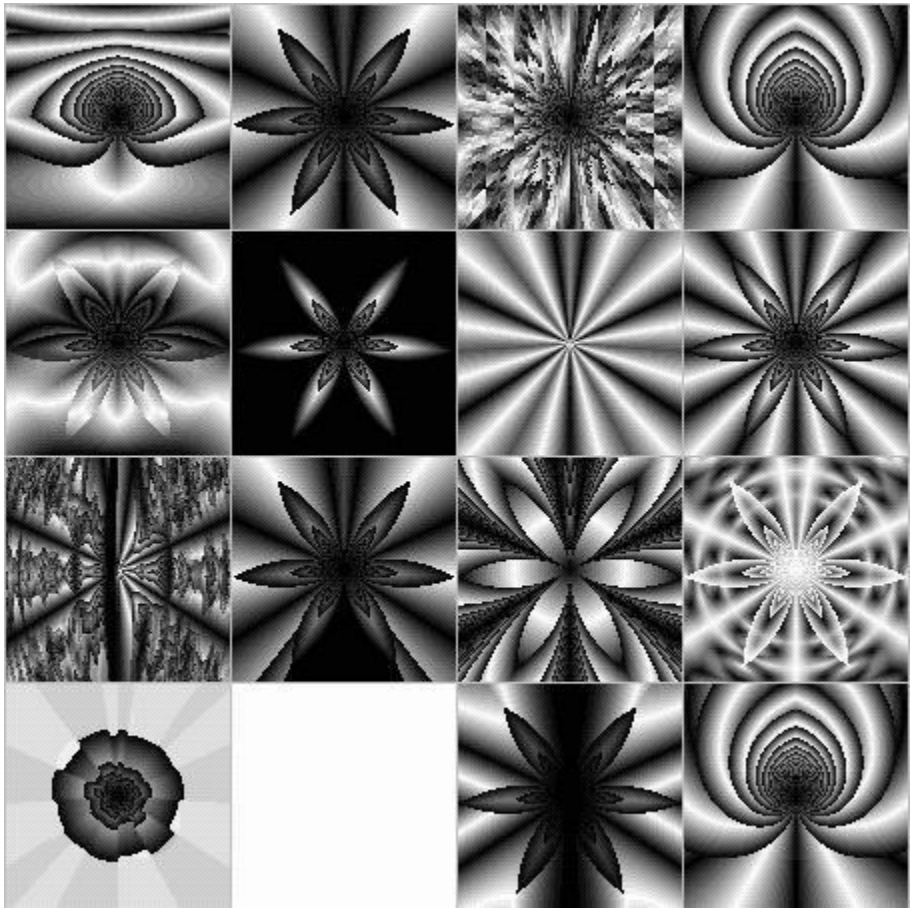


Exemple de mutatie

(triwave (mod (triwave (PHY)) (RAD)))



(triwave (mod (lerp (PHY) (Y) (Y)) (RAD)))

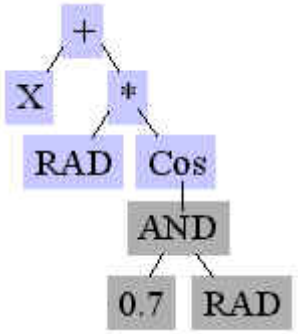


Exemple de mutatie

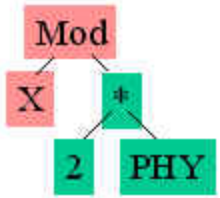
1. (triwave (mod (lerp (PHY) (Y) (Y)) (RAD)))
2. (mod (triwave (PHY)) (RAD))
3. (triwave (mod (^ (triwave (PHY)) (X)) (RAD)))
4. (triwave (mod (PHY) (RAD)))
5. (triwave (lerp (mod (triwave (PHY)) (RAD)) (Y) (RAD)))
6. (triwave (mod (max (triwave (PHY)) (RAD)) (RAD)))
7. (triwave (sin (triwave (PHY))))
8. (triwave (mod (+ (triwave (PHY)) (RAD)) (RAD)))
9. (triwave (mod (triwave (PHY)) (warped_color_noise (RAD) #(0.653 0.865 -0.369) #(0.683 0.337 -0.625) (X))))
10. (mod (mod (triwave (PHY)) (RAD)) (PHY))
11. (triwave (mod (RAD) (triwave (PHY))))
12. (triwave (rotate (mod (triwave (PHY)) (RAD)) (RAD)))
13. (triwave (mod (gradient (triwave (PHY))) (RAD)))
14. (invert (triwave (mod (triwave (PHY)) (RAD))))
15. (triwave (* (mod (triwave (PHY)) (RAD)) (X)))
16. (triwave (mod (abs (PHY)) (RAD)))

Combinare

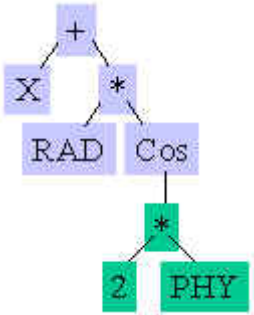
Parinte 1



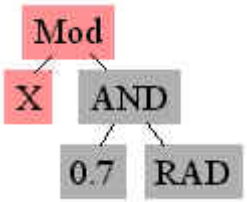
Parinte 2



Copil 1



Copil 2



Exemplu de combinare

Parinte 1:

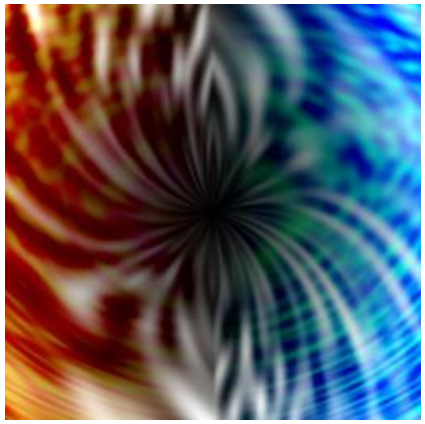
(lerp (* (vector -0.422 (warped_bw_noise (RAD) #(-0.468 -0.375 -0.624) (Y) (RAD)) (RAD)) (X)) (RAD) (bw_noise (PHY) (log 0.529)))

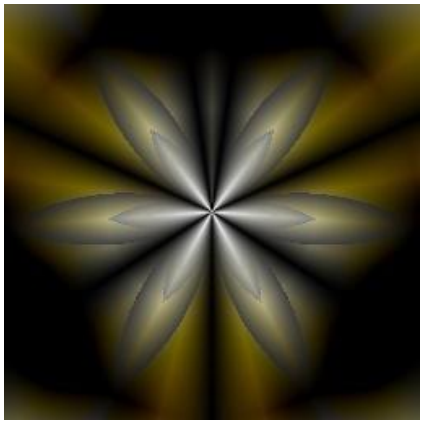
Domeniu: [-1..1] x [-1..1]

Parinte 2:

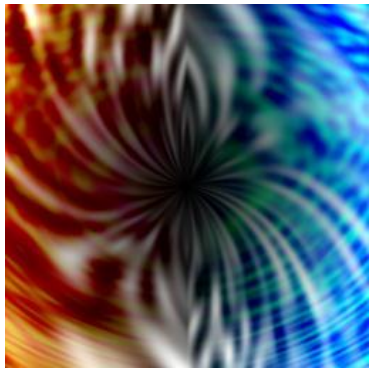
(lerp (triwave (PHY)) (* (lerp #(0.524 0.389 -0.394) (- (triwave (RAD)) 0.590) (PHY)) (triwave (mod (triwave (PHY)) (RAD)))) (RAD))

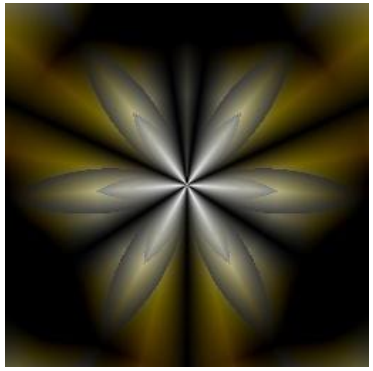
Domeniu: [-1..1] x [-1..1]

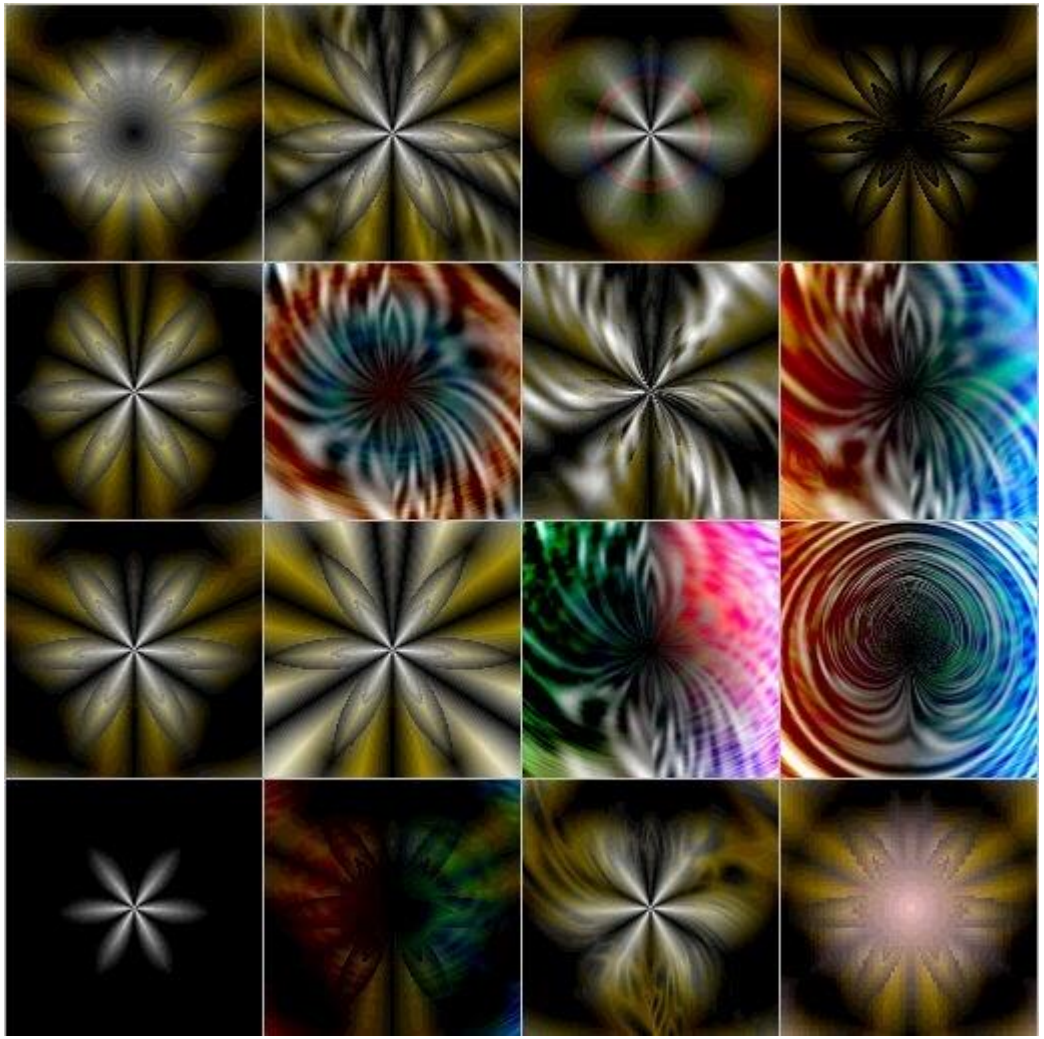




Exemplu de combinare - copii





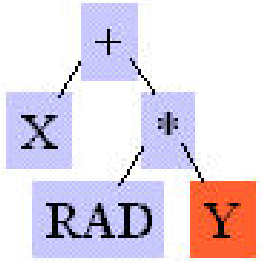


Tranzitii intre imagini

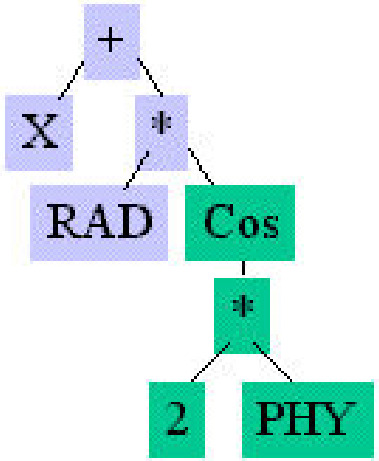
- Utilizatorul selecteaza o imagine sursa si o imagine destinatie
- Sistemul Gaia realizeaza automat tranzitia intre imagini
- Daca **genotipuri** (relativ) **similare** – efecte imteresante
- Daca **genotipuri diferite** – simpla amestecare
- Genotipuri relativ similare daca nodurile superioare sunt la fel
- Genotipul nou generat este *parametrizat in functie de timp*
- Noul genotip are noduri comune intre sursa si destinatie si interpoleaza in timp noduriel care sunt diferite
- t=0 genotip = sursa
- t=1 genotip = destinatie
- Intre 0 si 1 genotipul – o combinatie intre sursa si destinatie

Tranzitii intre imagini

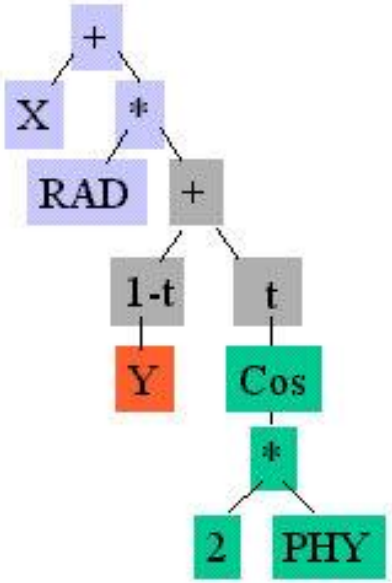
Genotip sursa

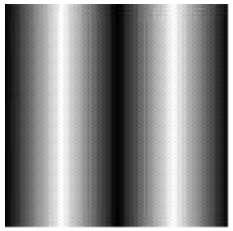


Genotip destinatie



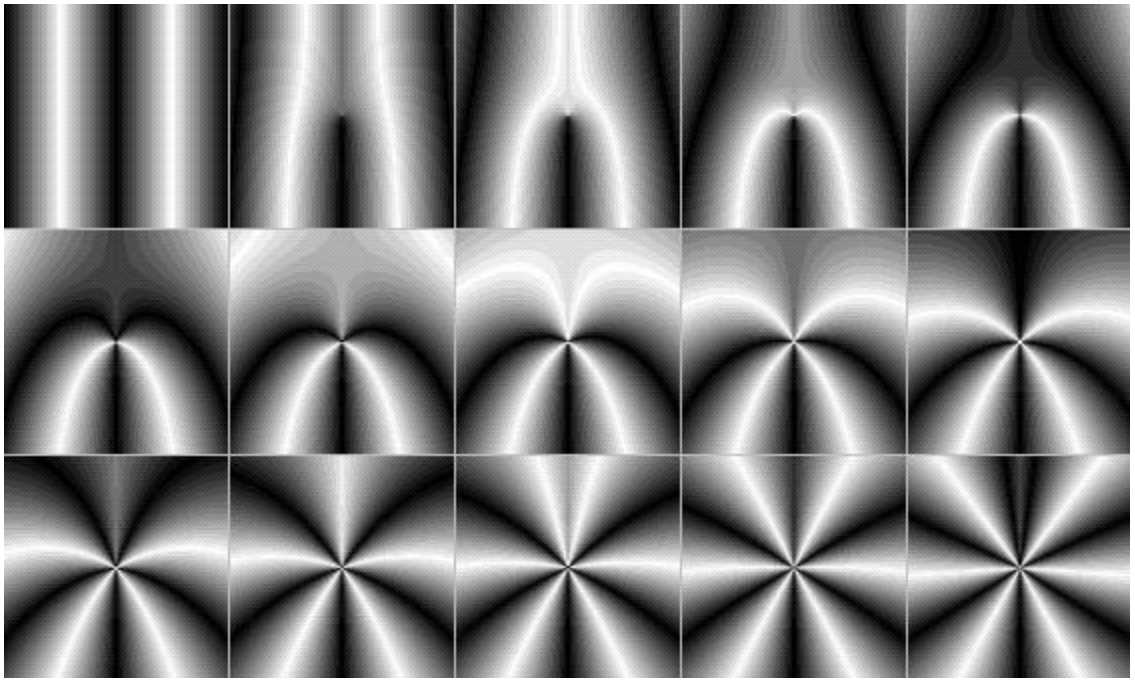
Genotip nou

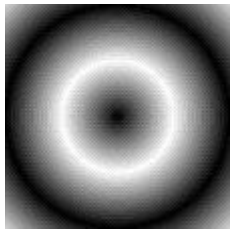




Tranzitii intre imagini

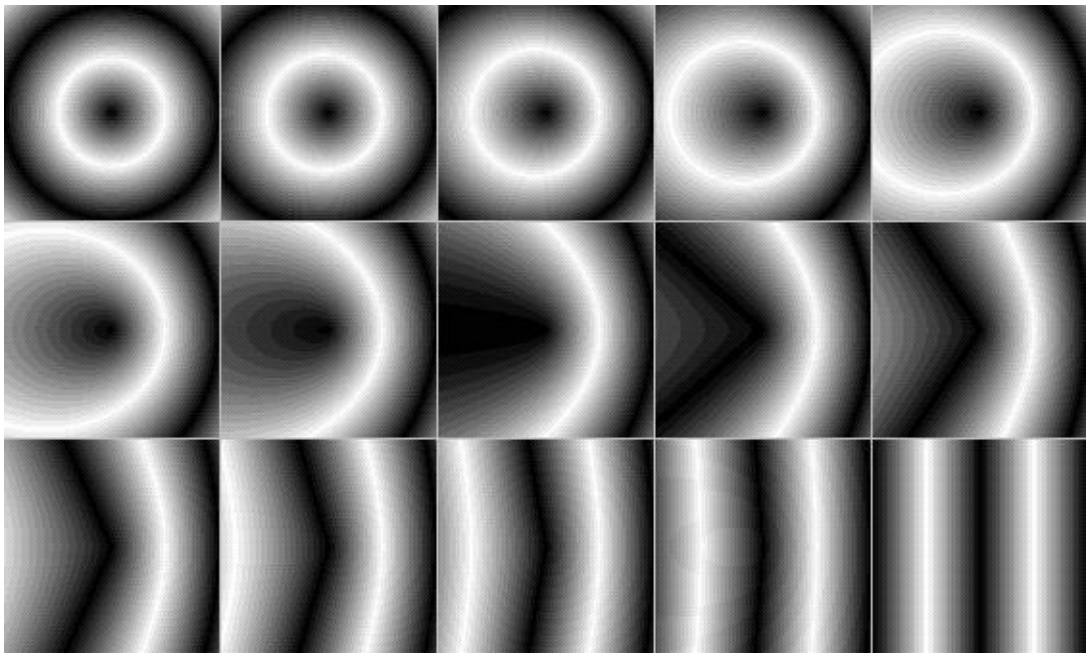
Sursa: (triwave (abs (X))) **Dest:** (triwave (&& (PHY) (PHY)))





Tranzitii intre imagini

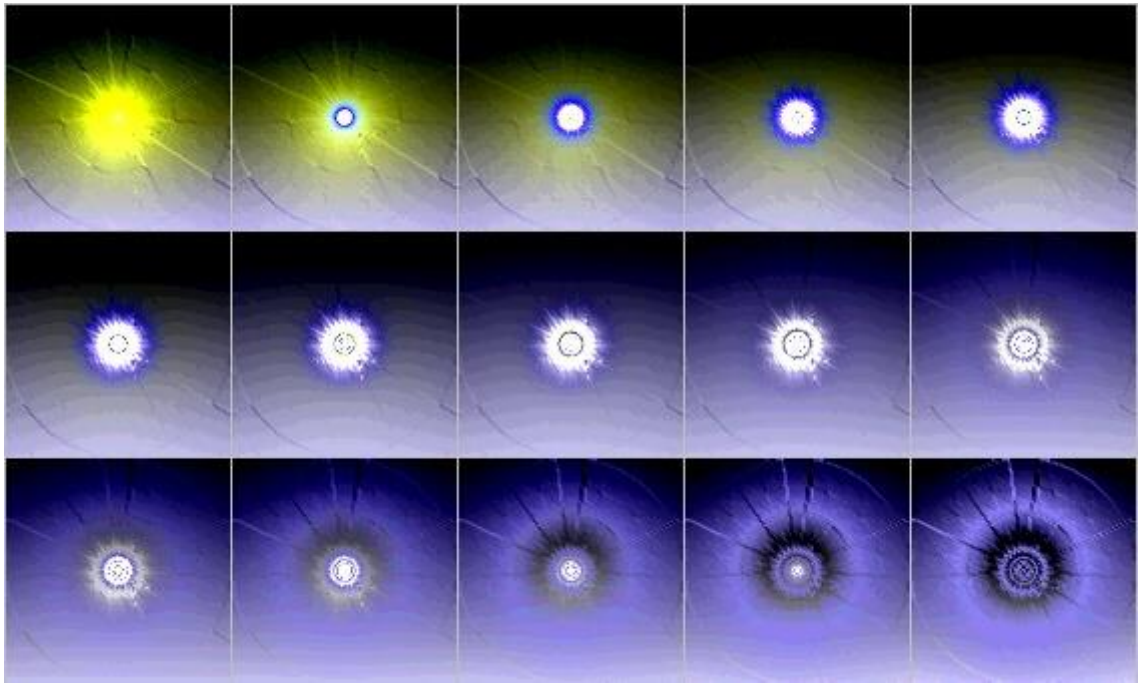
Sursa:(triwave (abs (RAD))) **Dest:**(triwave (abs (X)))



Tranzitii intre imagini

Sursa: (lerp (/ (gradient (& (PHY) (RAD))) (RAD)) (Y) #(0.545 0.495 0.981))

Dest: (lerp (Y) #(0.545 0.495 0.981) (/ (gradient (& (PHY) (RAD))) (RAD)))



Invatare automata

Universitatea Politehnica Bucuresti

Anul universitar 2021-2022

Adina Magda Florea

Curs Nr. 11

1. **Predictie Bayesiana**
2. **Aspecte teoretice ale invatarii - Invatare PAC**

1. Predictie Bayesiana

- Utilizarea teoriei probabilitatilor in predictie
- Modelul de invatare: date si ipoteze de invatare
- In predictia Bayesiana: datele sunt instante ale unor variabile aleatoare iar ipotezele sunt modele ale comportarii domeniului

Regula lui Bayes

- Predictia Bayesiana calculeaza probabilitatea fiecarei ipoteze pe baza datelor obtinute si face predictii pe baza acestora
- Predictia se face utilizand **toate ipotezele**, nu numai pe cele mai bune ipoteze

Regula lui Bayes

$$\mathbf{P}(b|a) = \mathbf{P}(a|b)\,\mathbf{P}(b) / \mathbf{P}(a)$$

Generalizare

$$\mathbf{P}(Y|X) = \mathbf{P}(X|Y)\,\mathbf{P}(Y) / \mathbf{P}(X)$$

cu normalizare

$$\mathbf{P}(Y|X) = \alpha\,\mathbf{P}(X|Y)\,\mathbf{P}(Y)$$

Predictie Bayesiana

Exemplu: Pungi cu bomboane

h1: 100% cirese

h2: 75% cirese 25% lamaie

h3: 50% cirese 50% lamaie

h4: 25% cirese 75% lamaie

h5: 100% lamaie

H- multimea de ipoteze, cu valori h1 .. h5 (tipul pungii de bomboane)

Se culeg probe d_1, d_2, … - variabile aleatore cu valori posibile cirese sau lamaie

Problema: prezice tipul de aroma a urmatoarei bomboane

Predictie Bayesiana

Fie **D** datele cu valorile observate **d** (lamaie sau cirese)

- Probabilitatea fiecarei ipoteze, pe baza regulii lui Bayes, este:

$$P(h_i|\mathbf{d}) = \alpha\, P(\mathbf{d}|h_i)\, P(h_i) \qquad i=1,5 \qquad (1)$$

$P(h3|d_1=\text{lamaie})$

- Elemente cheie: ipotezele apriori (**prior**) $P(h_i)$ si probabilitatea (**likelihood**) unei probe pentru fiecare ipoteza $P(\mathbf{d}|h_i)$

$$P(\mathbf{d}|h_i) = \Pi_j\, P(d_j|h_i) \qquad (2)$$

- Predictia asupra unei ipoteze necunoscute X

i.i.d.

$$\mathbf{P}(X|\mathbf{d}) = \Sigma_i\, \mathbf{P}(X|d,h_i)\, P(h_i|\mathbf{d}) = \Sigma_i\, \mathbf{P}(X|h_i)\, P(h_i|\mathbf{d}) \qquad (3)$$

Probabilitatea apriori $P(h_i)$ este cunoscuta

$P(d_2=\text{lamaie}|d_1=\text{lamaie})$

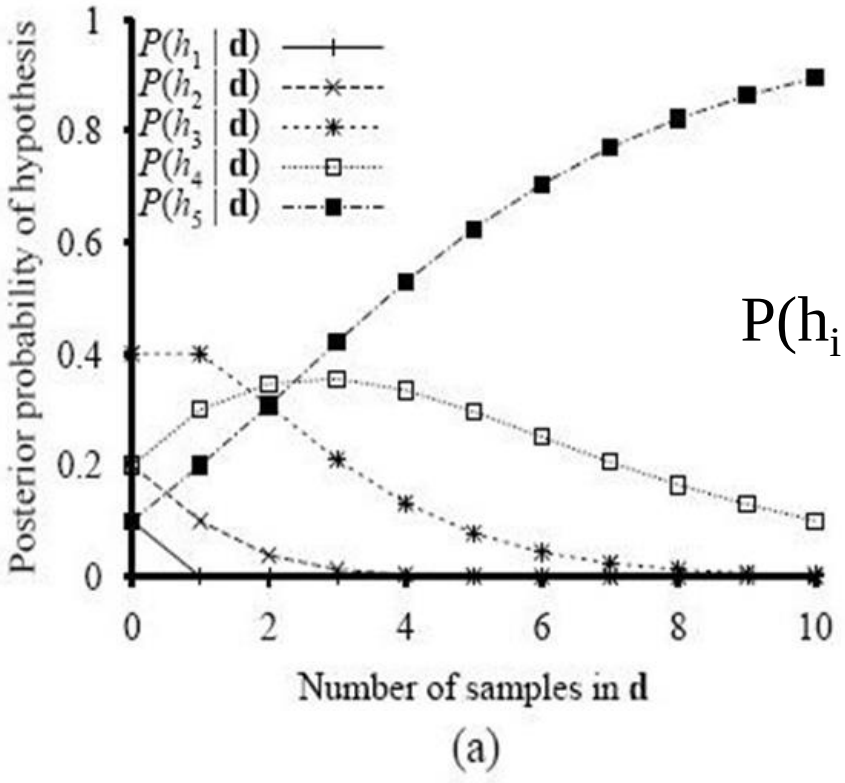
h1	h2	h3	h4	h5
0.1	0.2	0.4	0.2	0.1

Predictie Bayesiana

Presupunem probabilitatea apriori pentru

h1	h2	h3	h4	h5
0.1	0.2	0.4	0.2	0.1

Evolutia valorilor probabilitatilor ipotezelor, care incepe cu valorile prior



$P(h_i|d_1,\ldots,d_{10})$ din ecuatia (1)

Presupun h5, extrag 10 b.

h1 h2 h3 h4 h5

0.1 0.2 0.4 0.2 0.1

$$P(h_i|\mathbf{d}) = \alpha\, P(\mathbf{d}|h_i)\, P(h_i) \quad (1)$$

h1: 100% cirese

h2: 75% cirese 25% lamaie

h3: 50% cirese 50% lamaie

h4: 25% cirese 75% lamaie

h5: 100% lamaie

Calculam P(hi|d_1=lamaie)

P(lamaie) = 0.1*0 + 0.2*0.25 + 0.4*0.5 + 0.2*0.75+ 0.1*1 = 0.5

α = 1/0.5 = 2

P(h1|lamaie) = α P(lamaie|h1)P(h1) = 2* (0*0.1) = 0

P(h2|lamaie) = α P(lamaie|h2)P(h2) = 2 * (0.25*0.2) = 0.1

P(h3|lamaie) = α P(lamaie|h3)P(h3) = 2 * (0.5*0.4) = 0.4

P(h4|lamaie) = α P(lamaie|h4)P(h4) = 2 * (0.75*0.2) = 0.3

P(h5|lamaie) = α P(lamaie|h5)P(h5) = 2 * (1*0.1) = 0.2

h1 h2 h3 h4 h5

0.1 0.2 0.4 0.2 0.1

$$P(hi|\mathbf{d}) = \alpha\, P(\mathbf{d}|hi)\, P(hi) \quad (1)$$

$$P(\mathbf{d}|h_i) = \Pi_j\, P(d_j|h_i) \quad (2)$$

h1: 100% cirese

h2: 75% cirese 25% lamaie

h3: 50% cirese 50% lamaie

h4: 25% cirese 75% lamaie

h5: 100% lamaie

Calculam $P(hi|d_1,d_2)$

P(lamaie,lamaie) = 0.1*0*0 + 0.2*0.25*0.25 + 0.4*0.5*0.5 + 0.2*0.75*0.75+ 0.1*1*1 = 0.325

α = 1/0.325 = 3.0769

P(h1|lamaie,lamaie) = α P(lamaie,lamaie|h1)P(h1) = 3* (0*0*0.1) =0

P(h2|lamaie,lamaie) = α P(lamaie,lamaie|h2)P(h2) = 3 * (0.25*0.25*0.2) = 0.0375

P(h3|lamaie,lamaie) = α P(lamaie,lamaie|h3)P(h3) = 3 * (0.5*0.5*0.4) = 0.3

P(h4|lamaie,lamaie) = α P(lamaie,lamaie|h4)P(h4) = 3 * (0.75*0.75*0.2) = 0.3375

P(h5|lamaie,lamaie) = α P(lamaie,lamaie|h5)P(h5) = 3 * (1*1*0.1) = 0.3

h1: 100% cirese

h2: 75% cirese 25% lamaie

h3: 50% cirese 50% lamaie

h4: 25% cirese 75% lamaie

h5: 100% lamaie

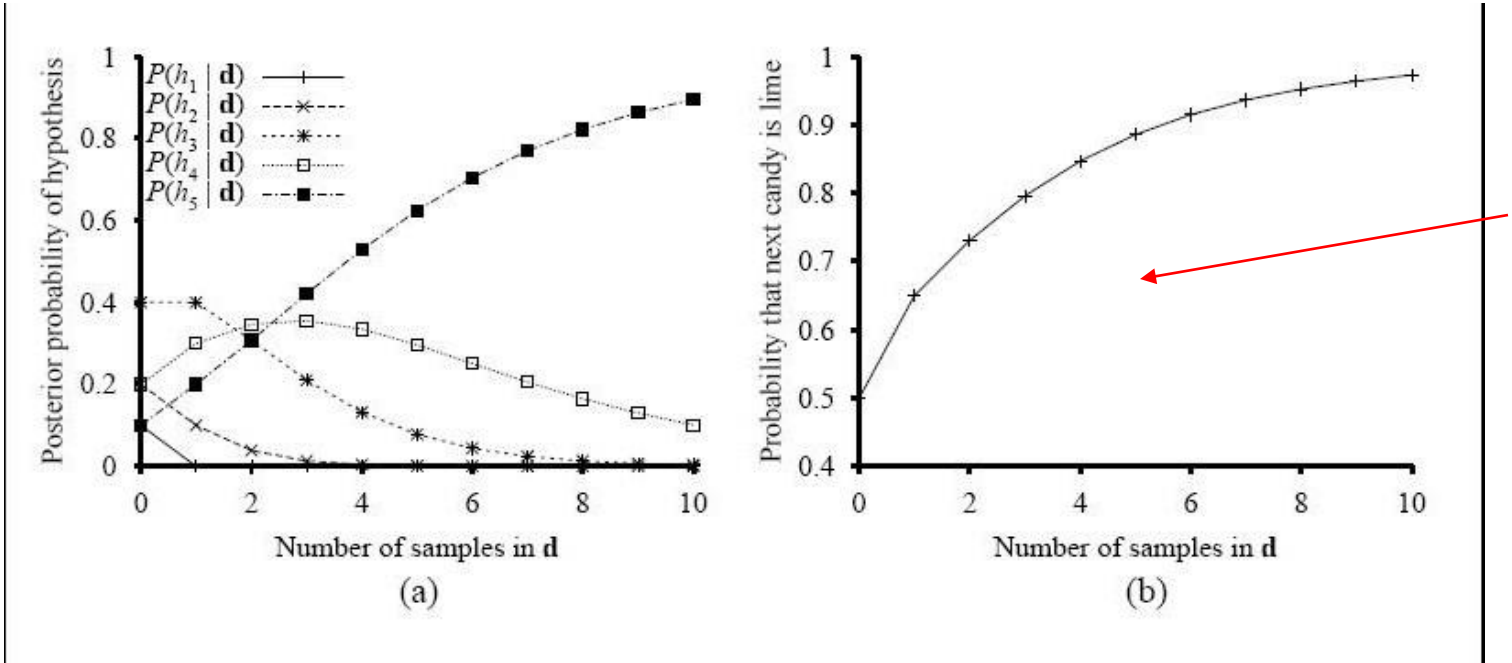
h1	h2	h3	h4	h5
0.1	0.2	0.4	0.2	0.1

$$\mathbf{P}(X|\mathbf{d}) = \Sigma_i \mathbf{P}(X|h_i) P(h_i|\mathbf{d}) \quad (3)$$

$P(d_2=\text{lamaie}|d_1=\text{lamaie}) = P(d_2|h1)*P(h1|d_1) + P(d_2|h2)*P(h2|d_1)$

$+ P(d_2|h3)*P(h3|d_1) + P(d_2|h4)*P(h4|d_1) + P(d_2|h5)*P(h5|d_1) =$

$= 0*0.1+0.25*0.2+0.5*0.4+0.75*0.3+1*0.2 = 0.65$



Predictie Bayesiana

Observatii

- Ipoteza adevarata va domina in final predictia
- Predictia Bayesiana este optima: fiind dat setul de ipoteze, orice alta predictie va fi corecta mai putin frecvent
- Probleme daca spatiul ipotezelor este mare
- **Aproximare - MAP**
  - Se fac predictii pe baza unei singure ipoteze, respectiv **ipoteza cela mai probabila,** respectiv

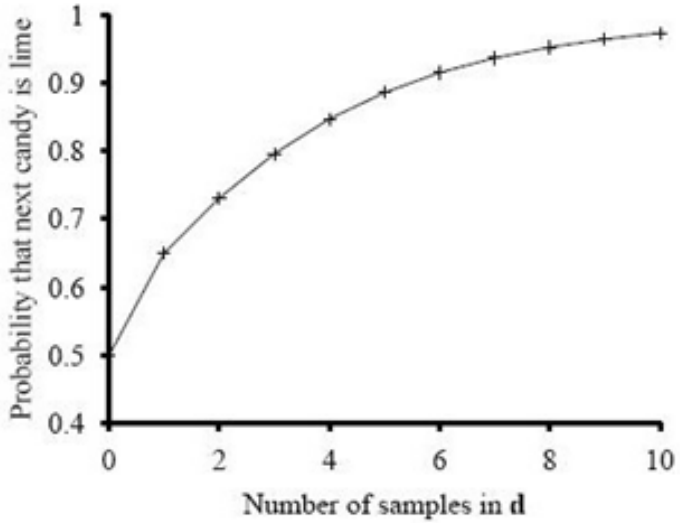
$$h_i \text{ care maximizeaza } P(h_i|\mathbf{d})$$

MAP Learning – maximum aposteriori – vezi IA

Predictiile facute cf ipotezei h_{MAP} sunt aproximativ Baysiene in masura in care $P(X|D) \approx P(X \mid h_{MAP})$

Observatii

- In exemplul dat h_{MAP}=h5 dupa extragerea a 3 bomboane lamaie la rand deci MAP prezice ca a 4-a bomboana este lamiae cu probabilitatea 1.0 – o predictie mai riscanta decat predictia Bayesiana de 0.8
- Pe masura ce se colecteaza mai multe date, MAP se apropie de predictia Bayesiana



Observatii

- Gasirea h_{MAP} este mai simpla d.p.v al calcului decat predictia Bayesiana
- Daca presupunem o distributie uniforma a probabilitatii apriori peste spatiul ipotezelor, atunci invatarea MAP revine la a alege h_i care maximizeaza $P(\mathbf{d}|h_i)$
- **Maximul Likelihood**

Maximul likelihood learning

- Invatarea paramterilor unui model probabilistic
- **Exemplu:** o punga de bomboane cu cirese si lamai in proportie necunoscuta
- Daca proportia de bomboane este apriori la fel (distributie uniforma a probabilitatii apriori) putem folosi ML
- 1 variabila aleatore, **Aroma**, cu valori *cirese*, *lamaie* si **probabilitate cirese θ**
- Extragem **N** bomboane, cu **c** cirese si **l**=N-c lamai

Conform $P(\mathbf{d}|h_i) = \Pi_j P(d_j|h_i)$ (2)

$$P(\mathbf{d}|h_\theta) = \Pi_{j=1,N} P(d_j|h_\theta) = \theta^c (1- \theta)^l$$

Maximul likelihood learning

$$P(\mathbf{d}|h_\theta) = \Pi_{j=1,N}\, P(d_j|h_\theta) = \theta^c\,(1-\theta)^l$$

Ipoteza Maximum Likelihood este data de valoarea lui θ care maximizeaza $P(\mathbf{d}|h_\theta)$

Log likelihood

$$L(\mathbf{d}|h_\theta) = \log P(\mathbf{d}|h_\theta) = \Sigma_{j=1,N}\log P(d_j|h_\theta) = c\log\theta + l\log(1-\theta)$$

Pt a aflax maxim θ derivam L in raport cu θ

$$\frac{dL(d|h_\theta)}{d\theta} = \frac{c}{\theta} - \frac{l}{1-\theta} = 0 \qquad \theta = \frac{c}{c+l} = \frac{c}{N}$$

Ipoteza h_{ML} spune ca proportia de cirese din punga este egala cu proportia din bomboanele extrase pana la un moment

Maximul likelihood learning

Metoda Maximum Likelihood parameter learning

- Expresie pentru likelihood a datelor ca o functie de parametrii
- Se calculeaza derivate log likelihood
- Gaseste parametrii care fac derivate zero, deci max loglikelihood, deci max likelihood

Probleme

Expresii complicate

Daca setul de date este mic si anumite evenimente nu au fost observate (de ex nici o bomboana cu cirese) ipoteza ML atribuie probabilutate 0 acestor evenimente

Alt exemplu

In punga avem bomboane cu staniol colorat rosu sau verde – ***Invelis (I)*** si avem si variabila ***Aroma (A)*** – *cirese sau lamaie*

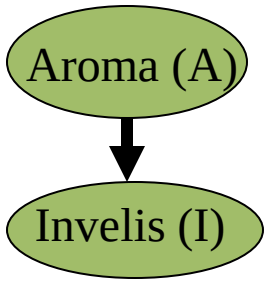
Invelis pt fiecare bomboana este ales probabilistic in functie de o distributie de probabilitate conditionate de aroma bomboanei

$P(A=\text{cireasa})$ - θ

$P(I=\text{rosu}|A)$ pt A=cireasa θ_1 si A=lamiae θ_2

In acest caz exista 3 parametrii θ θ_1 θ_2

P(A=cir)
θ



A	P(I|A)
cireasa	θ_1
lamaie	θ_2

Retea Bayesiana

Maximum Likelihood Learning

Likelihood ca o bomboana cu cirese sa fie ambalata in verde este

$P(A=\text{cirese}, I=\text{verde} \mid h_{\theta,\theta 1,\theta 2}) =$

$P(A=\text{cirese} \mid h_{\theta,\theta 1,\theta 2})\, P(I=\text{verde} \mid A=\text{cirese}, h_{\theta,\theta 1,\theta 2}) = \theta\,(1-\theta_1)$

- Extragem N bomboane din care **c** sunt cirese si **l** lamai
- $\mathbf{r_c}$ cirese au invelis rosu, $\mathbf{g_c}$ cirese au invelis verde, $\mathbf{r_l}$ lamai au invelis rosu, $\mathbf{g_l}$ lamai au invelis verde
- Likelihood este data de

$$P\left(d \middle| h_{\theta,\theta 1,\theta 2}\right) = \theta^c (1-\theta)^l \theta_1^{r_c} (1-\theta_1)^{g_c} \theta_2^{r_l} (1-\theta_2)^{g_l}$$

- Log likelihood

$$L=[c \log \theta + l \log(1-\theta)] + [r_c \log \theta_1 + g_c \log(1-\theta_1)] + [r_l \log \theta_2 + g_l \log(1-\theta_2)]$$

Cei trei parametrii θ θ_1 θ_2 apar fiecare intr-o singura suma deci pot fi optimizati separat

Maximum Likelihood Learning

$$\frac{\partial L}{\partial \theta} = \frac{c}{\theta} - \frac{l}{1-\theta} = 0$$

$$\theta = \frac{c}{c+l}$$

$$\frac{\partial L}{\partial \theta_1} = \frac{r_c}{\theta_1} - \frac{g_c}{1-\theta_1} = 0$$

$$\theta_1 = \frac{r_c}{r_c+g_c}$$

$$\frac{\partial L}{\partial \theta_2} = \frac{r_l}{\theta_2} - \frac{g_l}{1-\theta_2} = 0$$

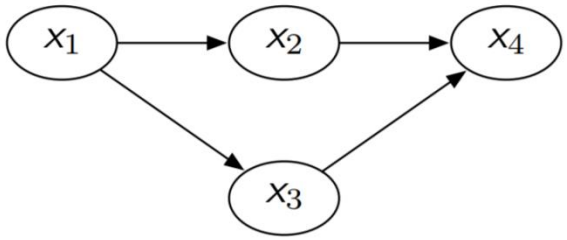
$$\theta_2 = \frac{r_l}{r_l+g_l}$$

Solutia pentru θ este aceeasi ca anterior, solutia pentru θ_1, probabilitatea ca o bomboana cu aroma de cirese sa aiba un invelis rosu este proportia de bomboane cirese cu invelis rosu observate

Maximum Likelihood in RB

- Cu date complete, invatarea parametrilor cu maximum likelihood pentru o retea bayesiana poate fi realizata prin descompunerea in sub-probleme, cate una pentru fiecare parametru
- De exemplu
- $\mathbf{p(x_1,\ldots, x_4) = p(x_1)p(x_2|x_1)p(x_3|x_1)p(x_4|x_2, x_3)}$

$$P(\mathbf{d}|h_\theta) = \Pi_{j=1,N}\, P(d_j|h_\theta) =$$
$$\Pi_{j=1,N} p(x_1)p(x_2|x_1)p(x_3|x_1)p(x_4|x_2, x_3)$$



Observatii

- Valorile parametrului pentru o variabila, fiind dati parintii ei, sunt reprezentate de valorile frecventei de aparitie a valorilor variabilei pentru fiecare valoare a variabilelor parinti
- Maximum likelihood Naive Bayes hypothesis
- Model de retea Bayesiana naiva – presupune independenta conditionala

Daca $x_1,..,x_n$ sunt independente conditional fiind dat y atunci
$P(x_1,..x_n|y) = \Pi_i P(x_i|y)$

Observatii

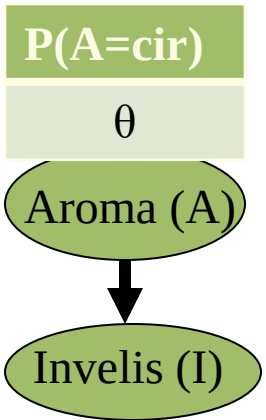
- Modelul prezentat anterior este naiv
- Presupunand variabile booleene, parametrii sunt

$\theta = P(C=true)$, $\theta_1 = P(X_i=true|C=true)$, $\theta_2 = P(X_i=true|C=false)$

Observand valorile atributelor x_1, x_2, ... probabilitatea fiecarei clase este

$$\mathbf{P}(C_k \mid x_1, x_2, \ldots) = \alpha\, \mathbf{P}(C_k) * \Pi_i\, \mathbf{P}(x_i|C_k)$$

si clasificarea $C\_MAP = \text{argmax}_k\ \mathbf{P}(C_k) * \Pi_i\, \mathbf{P}(x_i|C_k)$



A	P(I|A)
cireasa	θ_1
lamaie	θ_2

2. Aspecte teoretice ale invatarii

- Cand putem considera ca un algoritm de invatare a produs o ipoteza care poate prezice valori corecte pentru intrari necunoscute?
- Cum putem afla daca o ipoteza ***h*** este apropiata de ***f*** daca nu il cunoastem pe ***f***?
- De cate exemple avem nevoie pentru a produce un ***h*** bun?

Computational learning theory

Premisa

- Orice ipoteza care este semnificativ gresita *va fi gasita cu o probabilitate mare* dupa un numar mic de exemple, deoarece va face clasificari incorecte
- Astfel, orice ipoteza care este *consistenta cu un numar suficient de mare de exemple* de invatare este putin plauzibil sa fie grav gresita, adica trebuie sa fie **probabil aproximativ corecta** (PAC)
- **PAC = Probably Aproximately Correct**

Invatare PAC

- Orice algoritm de invatare care produce ipoteze care sunt PAC se numeste **algoritm de invatare PAC**
- Premisa invatarii PAC **este corecta** daca elementele din setul de invatare S si setul de test sunt **i.i.d.**
- Eroarea unei ipoteze ***h*** fata de ***f***, fiind data distributia D peste exemple, este:

$$\text{eroare}(h) = P(h(x) \neq f(x) \mid x \text{ din } D)$$

Probably Approximately Correct

- O ipoteza ***h*** este **aproximativ corecta** daca

  $$\text{eroare}(h) \leq \varepsilon$$

- Exista posibilitatea sa nu putem fixa o astfel de limita (S nereprezentativa)
- Dorim sa controlam probabilitatea ca ***h*** sa fie aproximativ corecta
- O ipoteza ***h*** este **probabil aproximativ corecta** daca

  $$P[\ \text{eroare}(h) \leq \varepsilon\ ] \geq 1-\delta$$

  $$(P[\ \text{eroare}(h) \geq \varepsilon\ ] \leq \delta)$$

Teorema PAC

- Daca exista un algoritm A care gaseste o ipoteza ***h*** $\in$ H consistenta cu N exemple de invatare, unde

$$N \geq \frac{1}{\varepsilon}(\ln(|H|) + \ln(\frac{1}{\delta}))$$

atunci **problema este "invatare PAC"** si

P[eroare(h) ≥ ε] ≤ δ

eroare(h) ≥ ε - h este "ε - *gresita*"

Demonstratie

- Dorim o limita pentru probabilitatea unei ipoteze care este consistenta si ε - *gresita* (eroare(h) $\geq \varepsilon$)
- Calculam probabilitatea ca o ipoteza grav gresita $h_B \in H_B$ sa fie consistenta cu N exemple
- Fie h_A consistenta cu N exemple de invatare

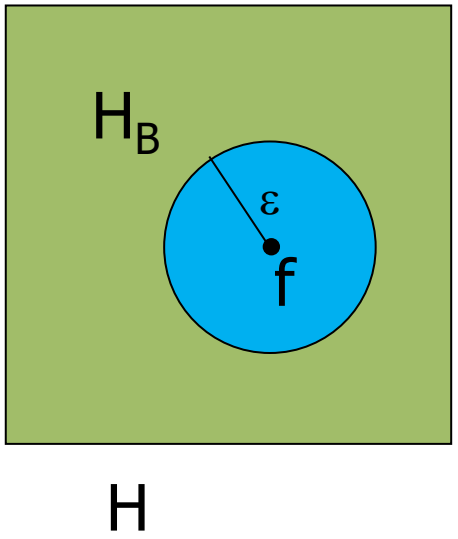
$P(h_A \text{ consistenta} \wedge h_A \ \varepsilon \text{ - } gresita) \leq$

$P(\exists h \in H: h \text{ cons} \wedge \varepsilon \text{ - } gresita) =$

$P(\exists h \in H_B: h \text{ cons})$

(in H_B eroare(h) $\geq \varepsilon$)

$P(\exists h \in H_B: h \text{ cons}) \leq \Sigma_{h \in HB} P(h \text{ cons})$



Demonstratie - cont

$$P(h \text{ cons}) = [P(h(x_1)=f(x_1)] \wedge \ldots [P(h(x_N))=f(x_N)] =$$

$$\Pi_{i=1,N} P(h(x_i)=f(x_i))$$

- **eroare(h_B)> ε** deci probabilitatea ca sa clasifice corect un exemplu (sa fie consistenta) este <1- ε
- Deci pentru N exemple, probabilitatea ca $\boldsymbol{h_B}$ sa fie consistenta este < $(1- \varepsilon)^N$

$$P(\exists h \in H_B: h \text{ cons}) \leq \Sigma_{h \in HB} P(h \text{ cons}) \leq |H_B| (1- \varepsilon)^N$$

$$\leq |H| (1- \varepsilon)^N$$

Demonstratie - cont

- Tinem cont de faptul ca $(1-x) \leq e^{-x}$

$$P(\exists h \in H_B: h \text{ cons}) \leq |H| (1-\varepsilon)^N \leq |H| e^{-\varepsilon N} \leq \delta$$

$$N \geq \frac{1}{\varepsilon}(\ln(|H|) + \ln(\frac{1}{\delta}))$$

$$N \geq \frac{0.69}{\varepsilon}(\log_2(|H|) + \log_2(\frac{1}{\delta}))$$

- Daca un algoritm de invatare produce o ipoteza care este consistenta cu multe exemple (N dat de formula de mai sus) atunci cu probabilitatea $1-\delta$ are eroarea cel mult ε
- N - Complexitatea de esantionare a spatiului de cautare

Exemplu

- Daca H este o multime de functii logice de ***n*** atribute – cate functii posibile de invatat?
- Daca avem 2^n elemente necesare pt a defini o functie, atunci

$$|H| = 2^{2^n}$$

- De ex cu 6 atribute |H| = nr cu 20 digiti
- Complexitatea de esantionare a spatiului creste proportional cu 2^n

Complexitatea de esantionare

$$| H | = 2^{2^n}$$

$$N \geq \frac{0.69}{\varepsilon}(2^n + \log_2(\frac{1}{\delta}))$$

Complexitatea de esantionare pt AD

Arbore de decizie, ***m*** atribute

H_k – numarul de AD cu adancimea k

$H_0=2$

H_{k+1} = (#arb atr rad) * (#arb stg posibili) * (#arb drept pos)=

$m*H_k*H_k$

Fie $L_k=\log_2 H_k$

$L_0=1$, $L_{k+1}=\log_2 m+2L_k$

rezulta $L_k=(2^k-1)(1+\log_2 m)+1$

$$N \geq \frac{0.69}{\varepsilon}((2^k-1)(1+\log_2 m)+1+\log_2(\frac{1}{\delta}))$$