BABEŞ-BOLYAI UNIVERSITY
Faculty of Mathematics and Computer Science

Inteligenţă Artificială

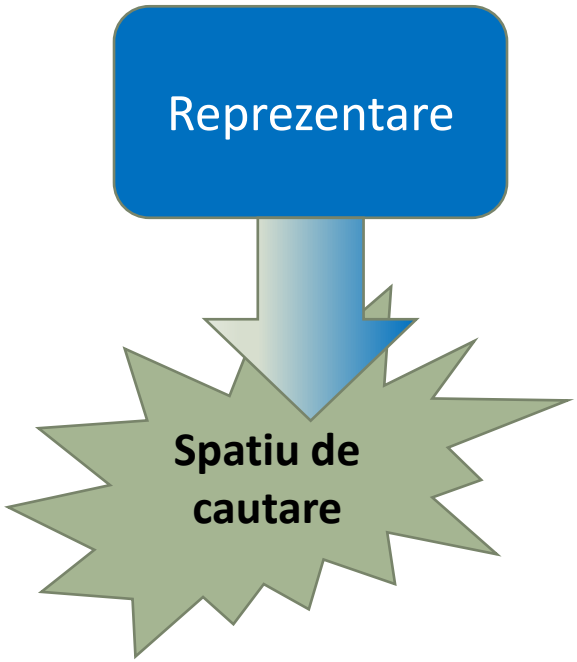
3: Simulated Annealing, Tabu Search

Camelia Chira

cchira@cs.ubbcluj.ro

Recap: Reprezentare

- Reprezentare comuna ex. SAT, TSP, NLP
- Alte posibilitati?



Reprezentare

- Reprezentare comuna ex. SAT, TSP, NLP
- Alte posibilitati?

NLP

- Sir binar de lungime kn, k=numarul de biti folositi pentru un numar real

$\langle b_{k-1} \dots b_0 \rangle$ → Transf. in x din [u,l] → $(\langle b_{k-1} \dots b_0 \rangle)_2 = (\sum_{i=0}^{k-1} b_i \cdot 2^i)_{10} = x'$

- Valoarea x reala corespunzatoare cu intervalul dat [u,l]

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

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

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

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

Precizia depinde de k

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

Reprezentare

- Reprezentare comuna ex. SAT, TSP, NLP
- Alte posibilitati?

TSP

- Permutare de **n** numere intregi;
- dar daca elementul **i** este un numar cuprins intre **1** si **n-i+1**, semnificand un oras ramas dintr-o lista de referinta **C**?

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

Reprezentare

(5 1 2 1 4 1 3 1 1)

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

Ruta

(1) Luam orasul nr. 5 din C **(5)** si apoi il stergem din C; C devine **(1 2 3 4 6 7 8 9)**
(2) Luam orasul nr. 1 din C **(1)** si apoi il stergem din C; C devine **(2 3 4 6 7 8 9)**
(3) Luam orasul nr. 2 din C **(3)** si apoi il stergem din C; C devine **(2 4 6 7 8 9)**
etc.

Reprezentare

- Reprezentare comuna ex. SAT, TSP, NLP
- Alte posibilitati?

SAT

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

Interpretare?

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

Metode clasice

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

- Cautare exhaustiva
- Cautare locala
- Metoda simplex

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

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

- Avem nevoie de algoritmi capabili sa iasa din optime locale

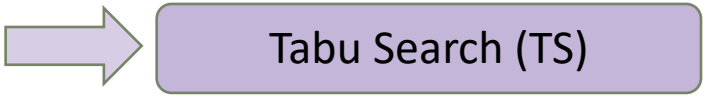
- Simulated Annealing
- Tabu Search

Metode de a iesi din optim local

Introducem in algoritm:

- Un parametru (numit ***temperatura***) care schimba probabilitatea de a ne muta dintr-un punct in altul al spatiului de cautare

- O ***memorie*** (istoric) care forteaza un algoritm sa exploreze noi zone ale spatiului de cautare



Local search (LS) – Simulated Annealing (SA)

```
procedure local search
begin
    x = some initial starting point in S
    while improve(x) ≠ 'no' do
        x = improve(x)
    return(x)
end
```

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

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

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

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

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

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

Local search (LS) – Tabu Search (TS)

```
procedure local search
begin
    x = some initial starting point in S
    while improve(x) ≠ 'no' do
        x = improve(x)
    return(x)
end
```

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

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

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

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

Iterated Hill-Climber (Iterated HC/ SAHC)

```
begin
    t = 0
    initialize best
    repeat
        local = FALSE
        select a current point c at random
        evaluate c
        repeat
            select all new points in the neighborhood of c
            select the point x from the set of new points
                with the best value of evaluation function eval
            if eval(x) is better than eval(c) then c = x
                                              else local = TRUE
        until local
        t = t+1
        if c is better than best then best = c
    until t = MAX
end
```

Numai aici iesim din optim local

Acest repeat returneaza **intotdeauna** un optim local

Stochastic HC

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

=> Stochastic Hill-Climber

begin

t = 0

select a current point ***c*** at random

evaluate ***c***

repeat

select the point x from the neighborhood of c

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

t = t+l

until $t = MAX$

end

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

Un punct nou selectat este acceptat cu o anumita probabilitate

Stochastic HC

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

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

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

Care este rolul parametrului T?

Sa presupunem ca avem o problema de maximizare:
eval(c) = 107
eval(x) = 120

eval(c)-eval(x) = -13 => x este mai bun decat c

Cu ce probabilitate este acceptat x pe baza lui T?

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

Stochastic HC

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

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

Alegerea valorii lui T este importanta

T=10 , eval(c) = 107

Daca x este mai slab p scade

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

Daca x este mai bun p creste

Probabilitatea de acceptare p in functie de x

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

Simulated Annealing (SA)

- Algoritmul SA este similar cu Stochastic HC
- Principala diferenta: **parametrul T se schimba in timpul rularii**
- SA incepe cu valori mari ale lui T (*random search la inceput*) si scade gradual valoarea lui T (*iterated HC la sfarsit*)
- In plus, SA accepta intotdeauna punctele mai bune decat cel curent

SA – analogie din termodinamica

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

a.k.a

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

SA Procedure

begin

$t = 0$

initialize T

select a current point $\boldsymbol{c}$ at random

evaluate $\boldsymbol{c}$

repeat

repeat

select a new point x from the neighborhood of c

if eval(c) < eval(x)

then $c \leftarrow x$

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

until (termination-condition)

$T \leftarrow g(T, t)$

$t \leftarrow t + 1$

until (halting-criterion)

end

SA checklist

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

$\Rightarrow$Structura spatiului de cautare si a vecinatatii, functia de evaluare

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

SA steps

PAS 1:

$T \leftarrow T_{max}$
select $\boldsymbol{c}$ aleator

PAS 2:

select x din vecinatatea lui c
if eval(x) e mai bun decat eval(c)
then select x ($c \leftarrow x$)
else select x cu probabilitatea $e^{\frac{eval(x)-eval(c)}{T}}$

repeat step 2 de k_T ori

PAS 3:

set T ←rT
if $T \geq T_{min}$ **then** goto PAS 2
else goto PAS 1

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

SA-SAT

```
procedure SA-SAT
begin
    tries ← 0
    repeat
        v ← random truth assignment
        j ← 0
        repeat
            if v satisfies the clauses then return v
            T = T_max · e^{-j·r}
            for k = 1 to the number of variables do
            begin
                compute the increase (decrease) δ in the
                    number of clauses made true if v_k was flipped
                flip variable v_k with probability (1 + e^{-δ/T})^{-1}
                v ← new assignment if the flip is m[illegible]e
            end
            j ← j + 1
        until T < T_min
        tries ← tries +1
    until tries = MAX-[illegible]
end
```

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

LS vs. SA-SAT

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

SA-SAT poate sa iasa din optime locale!

- Se incearca diferite valori TRUE/FALSE in v pentru fiecare variabila (*flip cu prob. p*)
- **p depinde de δ (improvement of flip) si T**

Parametrii SA-SAT

r = rata de scadere a temperaturii

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

Valori folosite (Spears, 1996):

$$T_{max} = 0.30$$
$$T_{min} = 0.01$$
$$r = \frac{1}{N \cdot tries}$$

```
procedure SA-SAT
begin
  tries ← 0
  repeat
    v ← random truth assignment
    j ← 0
    repeat
      if v satisfies the clauses then return v
      T = T_max · e^(-j·r)
      for k = 1 to the number of variables do
      begin
        compute the increase (decrease) δ in the
          number of clauses made true if v_k was flipped
        flip variable v_k with probability (1 + e^(-δ/T))^(-1)
        v ← new assignment if the flip is made
      end
      j ← j + 1
    until T < T_min
    tries ← tries + 1
  until tries = MAX-TRIES
end
```

TSP: Simulated Annealing

- SA in care
  - c = ruta
  - eval(c) = lungimea drumului pe ruta c

```
begin
    t = 0
    initialize T
    select a current point c at random
    evaluate c
    repeat
        repeat
            select a new point x from the neighborhood of c
            if eval(c) < eval(x)
                    then c ← x
                    else if random[0,1) < e^((eval(x)−eval(c))/T) then c ← x

        until (termination-condition)
        T ← g(T, t)
        t ← t + 1
    until (halting-criterion)
end
```

TSP: Simulated Annealing

Diferentele dintre diferitele abordari SA pentru TSP vin din:

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

TSP: Simulated Annealing
EXEMPLU PSEUDOCOD

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

NLP: Simulated Annealing

- SA este usor de aplicat in optimizare numerica
- Variabile continue => vecinatatea poate fi definita pe baza unei distributii Gauss (pentru fiecare variabila)

- Punct curent x

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

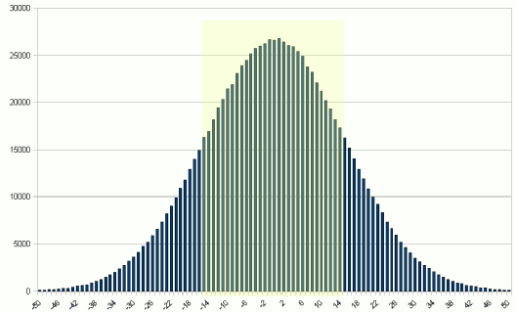
$$l_i \leq x_i \leq u_i, 1 \leq i \leq n$$

- Vecinul x' a lui x este:

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

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

$N(0, \sigma_i)$ este aleator ales din distributia Gauss (mean 0 , deviatie standard σ_i)



NLP: Simulated Annealing

- Schimbarea in functia de evaluare poate fi calculata:

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

- Daca $\Delta eval > 0$ (problema de minimizare):
  - Noul punct x' se accepta ca noua solutie
  - Altfel, x'este acceptat cu probabilitatea $e^{\frac{\Delta eval}{T}}$

Decizii SA pentru orice problema de optimizare numerica:

- Solutia initiala
- Vecinatate
- Selectarea unui vecin

-> usor de luat (i.e. random, Gaussian)

- Metoda de a modifica temperatura
- Conditia de terminare inner-loop
- Conditia de oprire a algoritmului
- Existenta unei faze postprocesare

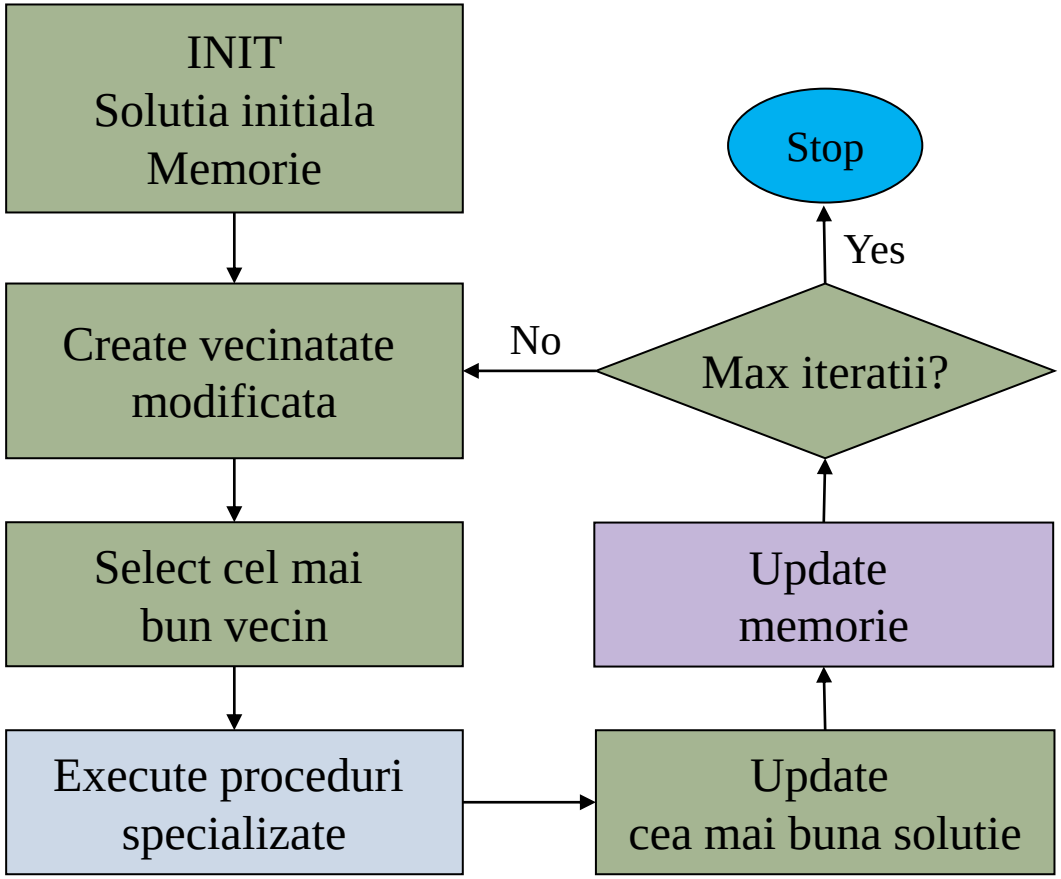
Tabu Search

- Idee principala: sa pastram o “memorie” care sa forteze cautarea sa exploreze noi zone ale spatiului de cautare

- Putem retine unele solutii examinate deja => ele devin puncte **tabu** (interzise) din spatiul de cautare

- Spre deosebire de SA, este o metoda determinista (fiind insa posibil sa ii adaugam elemente probabiliste)

Tabu Search


INIT
Solutia initiala
Memorie
Create vecinatate modificata
Select cel mai bun vecin
Execute proceduri specializate
Update cea mai buna solutie
Update memorie
Max iteratii?
No
Yes
Stop

Tabu Search

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

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

while stopping criterion is not fulfilled **do**

begin

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

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

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

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

end

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

SAT: Tabu Search

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

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

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

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

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

N(x):

xn_1=(**1**,1,1,1,0,0,0,1)
xn_2=(0,**0**,1,1,0,0,0,1)
xn_3=(0,1,**0**,1,0,0,0,1)
xn_4=(0,1,1,**0**,0,0,0,1)
xn_5=(0,1,1,1,**1**,0,0,1)
xn_6=(0,1,1,1,0,**1**,0,1)
xn_7=(0,1,1,1,0,0,**1**,1)
xn_8=(0,1,1,1,0,0,0,**0**)

SAT: Tabu Search

x=(0,1,1,1,0,0,0,1)
eval(x) = 27
Sa presupunem ca flip bit 3 => cea mai buna evaluare i.e. 31

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

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

$$\boldsymbol{M(i) = j, (cand\ j \neq 0)}$$

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

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

SAT: Tabu Search

x=(0,1,1,1,0,0,0,1)
eval(x) = 27
Sa presupunem ca flip bit 3 => cea mai buna evaluare i.e. 31

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

$$\boldsymbol{M(i) = j, (cand\ j \neq 0)}$$

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

In exemplul nostru, dupa schimbarea bitului 3:

- x=(0,1,**0**,1,0,0,0,1)
- *eval(x) = 31*
- *M =*

0	0	5	0	0	0	0	0

M(3)=5 ⇔ pentru urmatoarele 5 iteratii pozitia 3 nu este disponibila (tabu)

SAT: Tabu Search

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

M=

3	0	1	5	0	4	2	0

M dupa 5 iteratii

- *Bitii 2, 5 si 8 sunt disponibili si pot fi schimbati oricand*
- *Bitul 1 nu este disponibil inca 3 iteratii*
- *Bitul 3 nu este disponibil inca 1 iteratie*
- *Bitul 4 a fost schimbat chiar in aceasta iteratie si nu va fi disponibil urmatoarele 5 iteratii, etc.*

✓ Cea mai recenta schimbare: *M(4)=5 (bitul 4 schimbat acum 5-5=0 iteratii)*
✓ Celelalte schimbari in ordine inversa: *M(6)=4, M(1)=3, M(7)=2, M(3)=1*

⇒ **Solutia curenta este x=(1,1,0,0,0,1,1,1)**
Prespunem ca *eval(x)=33*

SAT: Tabu Search

x=(1,1,0,0,0,1,1,1)
eval(x)=33

M=

3	0	1	5	0	4	2	0

N(x):

xn_1=(**0**,1,0,0,0,1,1,1)
xn_2=(1,**0**,0,0,0,1,1,1)
xn_3=(1,1,**1**,0,0,1,1,1)
xn_4=(1,1,0,**1**,0,1,1,1)
xn_5=(1,1,0,0,**1**,1,1,1)
xn_6=(1,1,0,0,0,**0**,1,1)
xn_7=(1,1,0,0,0,1,**0**,1)
xn_8=(1,1,0,0,0,1,1,**0**)

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

Iteratia 6:

- Nu este permis sa schimbam bitii 1, 3, 4, 6 si 7
- Solutiile vecine obtinute din flips interzise sunt tabu

⇒ ***Solutia urmatoare este selectata din schimbari poz. 2, 5, 8***

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

⇒ *M=*

2	0	0	4	5	3	1	0

SAT: Tabu Search

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

Aspiration Criterion

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

Tabu Search: flexibilitate

Alte modalitati de face cautarea mai flexibila:

- Selectia determinista sa fie modificata intr-un probabilista (in care solutii mai bune sa aiba sanse mai mari sa fie selectate)

- Schimbarea orizontului memoriei: uneori este util sa fie mai mare, alteori e mai bine sa fie mai mic (e.g. cand este in hill-climbing area)

- Introducerea unei memorii de lunga durata!

TS cu long-term memory

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

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

$$\boldsymbol{H(i) = j}$$

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

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

5	7	11	3	9	8	1	6

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

TS cu long-term memory

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

SAT: TS cu long-term memory

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

eval(x') - penalty(x')

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

H=

5	7	11	3	9	8	1	6

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

Tabu Search

- Exista multe alte optiuni ce pot fi folosite in TS

- **Aspiration by default**:
  - Cand trebuie aleasa o solutie tabu, sa fie selectata dintre cele mai "vechi" considerate

- **Aspiration by search direction**:
  - Se memoreaza nu numai un set de miscari recente ci si daca acestea au insemnat sau nu o imbunatatire a solutiei

- **Aspiration by influence**:
  - *Influence* masoara gradul de schimbare a unei solutii noi (e.g. distanta dintre solutia noua si cea veche)
  - O miscare noua are o influenta mai mare daca s-a facut un pas mai mare de la vechea solutie la noua solutie

TSP: Tabu Search

- 8-city TSP

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

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

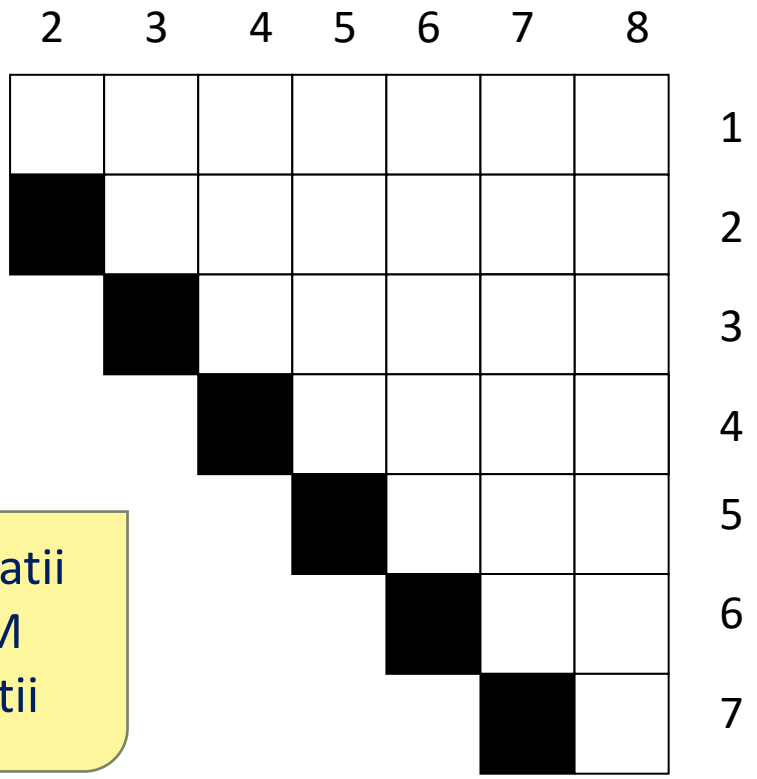
Vecinatate: un vecin obtinut prin interschimbarea a 2 orase (swap)
28 solutii in vecinatate: $\exists \binom{8}{2} = \frac{7 \cdot 8}{2 \cdot 1} = 28$ perechi de orase ce pot fi interschimbate

Care poate fi atunci structura memoriei?

TSP: Tabu Search

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

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



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

TSP: Tabu Search

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

M

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

H

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

TSP: Tabu Search

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

M

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

M(2,6)=5 indica cel mai recent swap
⇒ solutia precedenta era (7,3,5,**2**,1,**6**,4,8)
⇒ *swap(2,6)* este tabu pentru 5 iteratii

Tabu list:
- *swap(1,4)*
- *swap(2,6)*
- *swap(3,7)*
- *swap(4,5)*
- *swap(5,8)*

Cea mai veche

Numai 5 swaps din 28 posibile sunt interzise (tabu)

TSP: Tabu Search

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

H

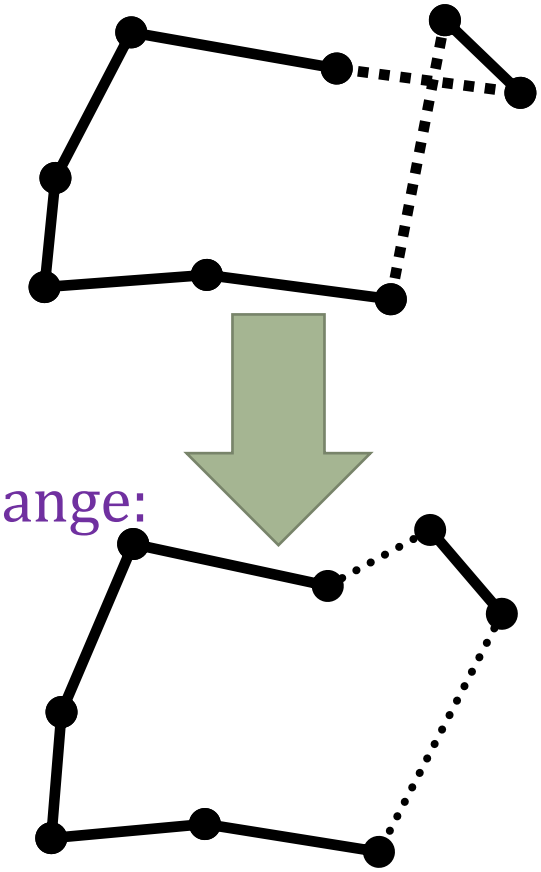
	2	3	4	5	6	7	8	
	0	2	3	3	0	1	1	1
		2	1	3	1	1	0	2
			2	3	3	4	0	3
				1	1	2	1	4
					4	2	1	5
						3	1	6
							6	7

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

TSP: Tabu Search

Rutele vecine pot fi determinate folosind alte metode

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



TSP: TS algorithm with 2-interchange moves

```
procedure tabu search
begin
    tries ← 0
    repeat
        generate a tour
        count ← 0
        repeat
            identify a set T of 2-interchange moves
            select the best admissible move from T
            make appropriate 2-interchange
            update tabu list and other variables
            if the new tour is the best-so-far for a given 'tries'
            then update local best tour information
            count ← count +1
        until count = ITER
        tries ← tries +1
        if the current tour is the best-so-far (for all 'tries')
        then update global best tour information
    until tries = MAX-TRIES
end
```

Parametri

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

Concluzii

Simulated Annealing	Tabu Search
Both designed to escape local optima	
✓ Can make uphill moves at any time	✓ Makes uphill moves only when it is stuck in local optima
✓ Stochastic algotihm	✓ Deterministic algorithm
✓ Lucreaza cu solutii complete (ca si algoritmii clasici de cautare exhaustiva si cautare locala) ✓ Pot fi opriti in orice iteratie	
▪ Au multi parametri de setat e.g. temperatura, rata de racire, memorie, etc.	

Next time...

Algoritmi Evolutivi