

# Simulated Annealing și Tabu Search

## I CERINȚE

---

- I. Să se implementeze algoritmul *Tabu Search* pentru problema rucsacului.
  - a. Generare soluție inițială și vecin
  - b. Algoritm și parametrizare
  - c. Experimente pe aceleași instanțe de la Tema I
  - d. Comparații cu rezultatele obținute de căutarea locală
  
2. Să se implementeze algoritmul *Simulated Annealing / Tabu Search* (la alegere) pentru problema TSP (instanța este indicată la laborator).
  - a. Generare soluție inițială și vecin
  - b. Algoritm și parametrizare
  - c. Experimente pe o instanță TSP din lista de mai jos

Observații (neîndeplinirea cerințelor din observații conduce la scăderea punctajului acordat):

- ✓ Instanța TSP considerată din lista de mai jos = aceeași cu numărul din grupă.
- ✓ Aplicația trebuie să fie modularizată, să permită parametrizarea algoritmului și afișarea soluției.
- ✓ Testați algoritmul în diverse variante pentru comparații pe instanța TSP primită.
- ✓ Rezultatele experimentelor trebuie salvate (cu indicarea setărilor folosite: algoritmul, valori parametri, număr rulări, calitatea soluției – best/avg).

## 2 TERMEN DE PREDARE

---

- **Lab 3**

**Total Punctaj Tema 2 = 150p**

### 3 PREDAREA TEMEI PRIN MS TEAMS

---

Incarcati urmatoarele fisiere **INAINTE** de a incepe lab-ul in care este setat termenul de predare:

1. O arhiva cu codul sursa
2. Un document (Word/PDF) care sa contina:
  - ✓ Descrierea pe scurt a algoritmului implementat (pseudocod) si principalelor componente (reprezentare solutie, functie de fitness, operatori, etc)
  - ✓ Indicarea parametrilor algoritmului
  - ✓ Tabele/grafice cu rezultatele obtinute (comparatii pentru cel putin 3 seturi de valori ale parametrilor pentru fiecare instanta de problema)
  - ✓ Analiza rezultatelor

### 4 PROBLEMA COMIS-VOIAJORULUI

---

Este una dintre cele mai cunoscute probleme de optimizare combinatorială cu multe aplicații (vezi <http://www.math.uwaterloo.ca/tsp/apps/index.html>).

Se consideră  $n$  orașe și un comis-voiajor care trebuie să viziteze toate orașele, trecând o singură dată prin fiecare și să se întoarcă în orașul inițial astfel încât costul total să fie cât mai mic.

Soluția poate fi reprezentată ca o permutare de  $n$  orașe. Costul unei soluții este suma distanțelor dintre orașe în ordinea în care apar în permutare.

„The simplicity of the statement of the problem is deceptive – the TSP is one of the most intensely studied problems in computational mathematics and yet no effective solution method is known for the general case. Indeed, the resolution of the TSP would settle the P versus NP problem and fetch a \$1,000,000 prize from the Clay Mathematics Institute.” [<http://www.math.uwaterloo.ca/tsp/problem/index.html>]

### 5 LISTA PROBLEME TSP

---

Instanțele TSP din lista de mai jos sunt disponibile în librăria online: <http://comopt.ifl.uni-heidelberg.de/software/TSPLIB95/tsp/>

Documentație: <http://comopt.ifl.uni-heidelberg.de/software/TSPLIB95/tsp95.pdf>

Listă probleme *Symmetric traveling salesman problem* (TSP):

1. eil101.tsp
2. eil176.tsp
3. berlin52.tsp
4. bier127.tsp
5. kroA100.tsp
6. kroB100.tsp
7. kroC100.tsp
8. kroD100.tsp
9. kroE100.tsp
10. lin105.tsp
11. pr76.tsp

- 12. p107.tsp
- 13. pr124.tsp
- 14. rat99.tsp
- 15. st70.tsp

## 6 SIMULATED ANNEALING

```
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 $\frac{eval(x)-eval(c)}{T}$  then c ← x
    until (termination-condition)
    T ← g(T, t)
    t ← t + 1
  until (halting-criterion)
end
```

- Vecinătatea lui c poate fi dată de 2-swap / 2-opt
- Schema de răcire dată de funcția g poate fi:
  - $T(k+1) = \alpha * T(k)$ , cu  $\alpha$  o valoare subunitară dar apropiată de 1 (de exemplu,  $\alpha=0.99$ )
  - $T(k+1) = T(k) / \log(k+1)$
  - $T(k+1) = T(k) / k$

unde  $T(k)$  este temperatura la iterația k.

Valoarea inițială a temperaturii trebuie să fie suficient de mare ex.  $T=10000$ .

- Criteriul de oprire poate fi:
  - Când valoarea temperaturii ajunge la o valoare minimă prestabilită
  - Numărul de iterații parcurse ajunge la un maxim
  - Valoarea atinsă de funcția de evaluare

## 7 TABU SEARCH

---

Un algoritm standard de tabu search este dat mai jos.

```
c = initSolution()
best = c
M = initMemory()
while (stop-criterion not met)
    x = getBestNeighbourNonTabu(c)
    updateMemory()
    c = x
    update best
return best
```

Condiția de oprire este dată de un număr maxim de iterații. Memoria reține numărul de iterații în care nu este permisă obținerea unui vecin prin 2-swap sau 2-opt pe anumite orașe.