Problema comis-voiajorului

Cautare locala, Cautare greedy, Hibridizari

1 CERINTE

Se considera problema comis voiajorului (travelling salesman problem – TSP) pentru care se cere:

1. Sa se implementeze un algoritm de *cautare locala* pentru TSP pe baza 2-opt sau 3-opt.
2. Sa se implementeze un algoritm de *cautare greedy* pentru TSP.
3. Sa se implementeze una din metodele urmatoare pentru TSP:
   a. Simulated Annealing
   b. Tabu Search

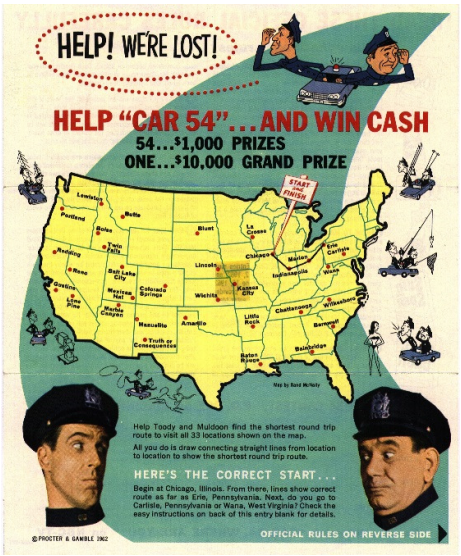
   Cerinte:
   - ✓ Specificati elementele algoritmului ales: pseudocod, parametri.
   - ✓ Studiati diferite schimbari posibile in algoritm si analizati comparativ performantele obtinute.
4. Scrieti un raport care sa contina rezultatele obtinute. Faceti o analiza comparativa a rezultatelor.

Obs.

- Metodele implementate se vor testa pe mai multe instante TSP din libraria disponibila online (http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/).
  Minim urmatoarele instante: eil51.tsp, eil76.tsp, eil101.tsp + 2 instante la alegere.
- Raportul trebuie sa aiba structura prezentata in cadrul laboratorului.

2 PROBLEMA COMIS-VOIAJORULUI

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



Se considera n orase si un comis-voiajor care trebuie sa viziteze toate orasele, trecand o singura data prin fiecare si sa se intoarca in orasul initial astfel incat costul total sa fie cat mai mic.

Solutia poate fi reprezentata ca o permutare de n orase. Costul unei solutii este suma distantelor dintre orase in ordinea in care apar in 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 $1,000,000prize from the Clay Mathematics Institute."

[http://www.math.uwaterloo.ca/tsp/problem/index.html]

3 CAUTARE LOCALA

1. Se alege o solutie din spatiul de cautare si se evalueaza. Aceasta este *solutia curenta*.

   Ex. Pentru initializare, poate fi generata aleator o permutare.

2. Se aplica o transformare solutiei curente pentru a genera o *noua solutie* care este evaluata.

   Transformarea 2-opt

   - Se aleg aleator doi indici i si j astfel incat 1<=i<j<=n
   - Se inverseaza ordinea elementelor din permutare cu indici cuprinsi intre i si j

3. Daca *solutia noua* este mai buna decat *solutia curenta* atunci se salveaza *noua solutie curenta*. Altfel, *solutia noua* se sterge.

4. Se repeta pasii 2 si 3 pana cand nici o transformare nu mai imbunatateste *solutia curenta*.

4 CAUTARE GREEDY

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

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

Aceasta abordare poate fi imbunatatita in mai multe feluri (vezi curs).

5 SIMULATED ANNEALING

- Vecinatatea lui c poate fi data de 2-opt
- Schema de racire data de functia g poate fi:

```
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
```

- T(k+1)= alpha * T(k), cu alpha o valoarea subunitara 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 iteratia k.

Valoarea initiala a temperaturii trebuie sa fie suficient de mare ex. T=10000.

- Criteriul de oprire poate fi:
  - Cand valoarea temperaturii ajunge la o valoare minima prestabilita
  - Numarul de iteratii parcurse ajunge la un maxim
  - Valoarea atinsa de functia de evaluare

6 TABU SEARCH

Tabu search porneste de la o solutie initiala pe care incearca sa o imbunatateasca evitand solutii incluse intr-o lista tabu.

$s \leftarrow$ GenerateInitialSolution()
$TabuList \leftarrow \emptyset$
while termination conditions not met **do**
  $s \leftarrow$ ChooseBestOf($\mathcal{N}(s) \setminus TabuList$)
  Update($TabuList$)
endwhile

Structura unui algoritm complet TS care foloseste un criteriu la care se aspira si memoria de lunga durata (un istoric mai lung al cautarii) este data mai jos.

$s \leftarrow$ GenerateInitialSolution()
InitializeTabuLists($TL_1, \ldots, TL_r$)
$k \leftarrow 0$
while termination conditions not met **do**
  $AllowedSet(s,k) \leftarrow \{s' \in \mathcal{N}(s) \mid s$ does not violate a tabu condition,
  or it satisfies at least one aspiration condition$\}$
  $s \leftarrow$ ChooseBestOf($AllowedSet(s,k)$)
  UpdateTabuListsAndAspirationConditions()
  $k \leftarrow k+1$
endwhile

O abordare de implementare Tabu Search folosind vecinatate 2-opt este prezentata in curs.

Printre modificarile aduse algoritmului intial TS puteti considera ‚aspiration criterion', memorie de lunga durata, strategii de diversificare (vezi curs).