Algoritmi metaeuristici.

Lab 2: Probleme de optimizare combinatorială.
Tehnici de căutare locală și globală care folosesc o singură soluție candidat:
Simulated Annealing
Tabu Search

---

1. Probleme de optimizare combinatorială.

Problemele de optimizare combinatorială se caracterizează prin faptul că spațiul de căutare este discret, însă de dimensiuni mari ceea ce exclude o explorare exhaustivă.

Două dintre cele mai cunoscute probleme de optimizare combinatorială, care au și o serie de aplicații practice, sunt:

- Problema comis-voiajorului (travelling salesman problem)
- Problema rucsacului (knapsack problem)

1. 1. Problema comis-voiajorului (Travelling Salesman Problem).

Este una dintre cele mai cunoscute probleme de optimizare combinatorială putând fi enunțată după cum urmează: "se consideră o mulțime de n orașe și un comis-voiajor care trebuie să viziteze toate orașele, trecând o singură dată prin fiecare și să se întoarcă în orașul de pornire astfel încât costul total al circuitului să fie cât mai mic". Din punct de vedere formal, în varianta clasică problema este echivalentă cu cea a găsirii unui circuit Hamiltonian de cost minim într-un graf complet.

Problema poate fi rezolvată exact (prin analiza tuturor circuitelor posibile) pentru valori mici ale numărului de orașe, însă cum numărul de circuite este (n-1)!/2, pentru valori mari ale lui n nu există metode exacte eficiente. Dealtfel, este cunoscut că TSP face parte din clasa problemelor NP-complete.

Problema este importantă atât din punct de vedere teoretic cât si din punct de vedere practic întrucât o serie de probleme concrete pot fi formulate ca TSP:

- Identificarea rutei optime pentru mijloace de transport (persoane, mărfuri)
- Generarea traseelor urmate de dispozitivele de producere a circuitelor integrate
- Secventierea unui genom (reconstruirea genomului pornind de la fragmente secvențiate între care există suprapuneri).

Diferite aplicații pot fi găsite la [http://www.tsp.gatech.edu/apps/index.html]

Există diferite variante ale problemei:

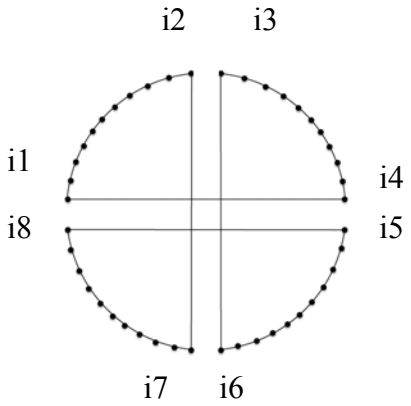
- Varianta asimetrică: costul trecerii de la un nod la altul depinde de sensul circuitului (Assymetric TSP)
- Variante cu restricții de precedență; restricțiile specifică faptul că un anumit nod trebuie vizitat înaintea altuia (Sequential Ordering Problem - SOP)
- Variante cu mai multe mijloace de transport care asigură aprovizionarea și care au fiecare o capacitate limitată (Capacitated Vehicle Routing Problem - CVRP)

- Variante generalizate în care există mai multe trasee directe între două noduri sau nodurile sunt înlocuite cu clustere de noduri (Generalized TSP)

Una dintre euristicile cel mai frecvent folosite este *euristica Lin-Kernighan* care se bazează pe ideea de a înlocui unele arce ale traseului cu altele în scopul reducerii costului. Cazul cel mai simplu este cel în care se înlocuiesc două arce ceea ce echivalează cu schimbarea ordinii de parcurgere a unei porțiuni din traseu (*transformarea 2-opt*). Euristica Lin-Kernighan constă în a găsi o secvență de transformări de tip 2-opt și se bazează pe a căuta (utilizând o strategie de tip backtracking) secvența cea mai profitabilă de transformări. Spațiul de căutare este limitat la un număr relativ mic (de exemplu 5) de variante pentru fiecare etapă de transformare.

Exemplu de transformare de tip 2-opt: In cazul a 6 orașe: A,B,C,D,E,F dacă traseul inițial este (A,C,B,E,F,D) prin inlocuirea arcului (A,C) cu arcul (A,F), a arcului (F,D) cu arcul (C,D) și inversarea ordinii de parcurgere a orașelor B și E se obține traseul (A,F,E,B,C,D). Se observă că acest traseu se poate obține din cel inițial prin inversarea ordinii în secvența (C,B,E,F).

O altă transformare utilizată de către euristicile pentru TSP este cea bazată pe 4 interschimbări numită tehnica "podului dublu" prin care un traseu de forma [i1..i2][i3..i4][i5..i6][i7..i8] este transformat într-un traseu de forma [i2..i1][i4..i3][i6..i5][i8..i7].



1.2.Problema rucsacului (Knapsack Problem)

Varianta clasică de formulare a problemei este: "Se consideră un set de n obiecte fiecare fiind caracterizat de o anumită dimensiune și o anumită valoare. Să se selecteze un subset de obiecte a căror dimensiune totală este mai mică sau cel mult egală cu capacitatea unui rucsac iar valoarea totală a obiectelor selectate este maximă."

Spațiul de căutare este reprezentat de toate submulțimile posibile ale mulțimii de obiecte deci are dimensiunea 2^n.

Probleme concrete care se pot reduce la problema rucsacului:

- Construirea unui portofoliu de investiții (scopul fiind maximizarea profitului fără a depăși suma disponibilă pentru investiții).
- Alocarea unei resurse (selecția task-urilor care pot utiliza resursa astfel încât să fie maximizat câștigul).
- Selecția mărfurilor ce urmează a fi plasate într-un container sau depozit.

Variante ale problemei:

- Cazul multicriterial: scopul nu este doar maximizarea unei valori ci optimizarea mai multor criterii
- Cazul multidimensional: dimensiunea unui obiect nu este specificată printr-o singură valoare ci prin mai multe
- Cazul rucsacelor multiple: se utilizează mai multe rucsace (este înrudită cu problema împachetării – "bin packing problem")

2. Metoda "Simulated Annealing" (SA)

2.1 **Descrierea metodei**

Simulated Annealing este o metaeuristică caracterizată de faptul că se acceptă, cu o anumită probabilitate, şi perturbări care conduc la soluții candidat mai puțin bune, iar această probabilitate descreşte în timp (depinzând de un parametru descrescător numit "temperatura" prin analogie cu procesul fizic denumit tratament termic sau călire).

Structura generală:

```
S=configurația inițială
T=valoarea inițială a parametrului "temperatură"
Repeat
   S'=perturb(S)
   If accept(S,S',T) then S=S'
   T=modifica(T)
Until <conditie de oprire>
```

Perturbarea depinde de problema de rezolvat iar condiția de acceptare este una probabilistă în care se ține cont de diferența de calitate dintre configurația curentă, S, şi cea perturbată, S'. O regulă simplă simplă de acceptare este:

```
Accept(S,S',T)
    If rand(0,1)<exp((f(s)-f(s'))/T) then
      Return True
    Else
      Return False
```

2.2. Rezolvarea problemei comis-voiajorului folosind "Simulated Annealing"

In rezolvarea unei probleme folosind metoda SA se parcurg următoarele etape:

a) *Alegerea modului de codificare a soluțiilor.* In cazul problemei comis-voiajorului codificarea naturală este cea de tip permutare: un traseu pentru n oraşe se specifică ca o permutare de ordin n. Această reprezentare asigură satisfacerea restricțiilor problemei (un oraş este vizitat exact o dată).

   - *Exemplu:* Dacă oraşele sunt numerotate astfel: 1-A, 2-B, 3-C, 4-D, 5-E, 6-F atunci traseul (A,C,B,E,F,D) corespunde permutării (1,3,2,5,6,4)

b) *Alegerea mecanismului de căutare locală (construirea unei noi configurații pornind de la cea curentă)*. Cea mai simplă variantă este cea bazată pe transformarea de tip 2-opt:
   - se aleg aleator doi indici i şi j astfel încât 1<=i<j<=n
   - se inversează ordinea elementelor din permutare ai căror indici sunt cuprinși între i şi j

   - *Exemplu:* Dacă traseul curent corespunde permutării (1,3,2,5,6,4) iar i=2 şi j=5 atunci permutarea obținută în urma transformării este: (1,6,5,2,3,4)

c) *Alegerea probabilității de acceptare a unei noi configurații.* Probabilitatea de acceptare a configurației C' obținută din transformarea C se poate calcula folosind distribuția Boltzmann:

   - $P(C'|C)=\min\{1,\exp(-(cost(C')-cost(C))/T(k)\}$

   - unde cost(C) este costul traseului C (adică valoarea funcției obiectiv care trebuie minimizată) iar T(k) este parametrul de control al procesului (temperatura).

d) *Alegerea schemei de răcire.* Daca se notează cu T(k) temperatura corespunzătoare iterației k atunci valoarea corespunzătoare iterației următoare se poate stabili în una dintre variantele:
   - $T(k+1)=T(0)/\log(k+1)$
   - $T(k+1)=T(0)/k$
   - $T(k+1)=a\ T(k)$, cu a o valoarea subunitara dar apropiată de 1 (de exemplu, a=0.99)

   Observație. Valoarea inițială a temperaturii (T(0)) trebuie să fie suficient de mare pentru a permite trecerea între oricare două configurații.

- *Stabilirea criteriului de oprire.* Criteriul de oprire se poate referi la valoarea temperaturii (se stabileşte o valoare minimă a temperaturii), la numărul de iterații parcurse (acest criteriu este corelat cu cel referitor la temperatură) sau la valoarea atinsă de funcția de optimizat (în cazul problemei comis-voiajorului este vorba de costul traseului).

Aplicație 1. Să se implementeze algoritmul Simulated Annealing pentru rezolvarea problemei comis-voiajorului. Pentru testare se pot folosi exemple de test de la http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/

Indicatie. O varianta de implementare este descrisă în SA_TSP.sci în care coordonatele punctelor sunt selectate manual.

Un exemplu de apel este: SA_TSP(10000,0.001)

Exerciții 1:

a) Să se implementeze o funcție pentru preluarea din fişier a coordonatelor punctelor (în acest caz se transmite numele fişierului ca parametru în funcția SA_TSP)
b) Să se analizeze comportarea algoritmului pentru problemele specificate in fişierele: eil51.tsp, eil76.tsp, eil101.tsp

c) Să se testeze comportarea algoritmului pentru fiecare dintre schemele de răcire specificate mai sus
d) Să se modifice funcția de calcul a costului astfel încât distanțele dintre puncte să fie calculate o singură dată (indicație: se vor reține distanțele dintre oricare două puncte într-o matrice)

2.3 Rezolvarea problemei rucsacului folosind "Simulated Annealing"

a) Codificarea soluției: vector binar
   - s_i=1 dacă obiectul i este selectat
   - s_i=1 dacă obiectul i nu este selectat
b) Perturbare locala: modificarea unei componente selectate aleator: $s_i = s_i - 1$
c) Evaluarea unei configurații: o variantă de a ține cont în evaluarea unei soluții candidat atât de funcția obiectiv (valoarea totală a obiectelor selectate) cât și de satisfacerea restricției (dimensiunea totală este mai mică decât capacitatea rucsacului) se bazează pe tehnica penalizării care combină funcția obiectiv și gradul de încălcare a restricției.

$$V(s) = \begin{cases} \sum_{i=1}^{n} v_i s_i & daca \sum_{i=1}^{n} w_i s_i \le c \\ \lambda \sum_{i=1}^{n} v_i s_i + (1-\lambda)(c - \sum_{i=1}^{n} w_i s_i) & daca \sum_{i=1}^{n} w_i s_i > c \end{cases}$$

Prin valoarea parametrului λ se poate controla importanța relativă a satisfacerii restricției respective a optimizării funcției obiectiv.

Aplicație 2. Să se implementeze algoritmul Simulated Annealing pentru rezolvarea problemei rucsacului.

Indicatie. O varianta de implementare este descrisă în SA_knapsack.

Exerciții 2:

a) Modificați implementarea de la Aplicatia 2 pentru a prelua datele de intrare din fișiere. Pentru date de test se va utiliza colecția de la http://people.sc.fsu.edu/~jburkardt/datasets/knapsack_01/knapsack_01.html

b) Testați funcția optim_sa din Scilab pentru problema comis-voiajorului și pentru problema rucsacului. Indicație

3. Tabu Search

3.1. Descrierea metodei

Tabu Search este un algoritm care folosește o căutare locală iterată caracterizată prin utilizarea unei liste de configurații deja vizitate care devin "interzise" (tabu) cel puțin pentru un anumit număr de iterații.

Structura generală a algoritmului Tabu Search:

```
S=configurația inițială
Best=S          // cea mai buna configurație
TabuList=[]    // lista de configuratii interzise este initial vida
TabuList=add(S,TabuList)   // adaugă S în lista
iter=1
Repeat
        S= perturb(S,TabuList)
        If better(S,Best) then Best=S;  endif
        iter=iter+1
Until iter<=iterMax // sau alta conditie de oprire
```

Perturbarea configurației curente se bazează pe identificarea în vecinătatea ei a unei configurații mai bune care nu se află în lista tabu. In momentul in care este aleasă o configurație perturbată aceasta se adaugă în lista tabu. Lista tabu este implementată ca o coadă cu dimensiune limitată (când este atinsă dimensiunea maximă a listei primul element din listă este eliminat).

```
Perturb(S,TabuList)
C=generateCandidate(S)     // element din vecinătatea lui S
// genereaza candidați și îl selectează pe cel mai bun care nu e in tabuList
For k=1:nrCandidates do
        C'= generateCandidate(S)      // generează un nou candidat
        If  C' (not in TabuList) and better(C',C) then C=C' endif
Endfor
If better(C,S) then S=C
                        If length(TabuList)>maxSize then TabuList=removeOld(TabuList)
                        TabuList=add(S,TabuList)
endif
Return S, TabuList
```

Prin better(S',S) se verifică dacă configurația S' este mai bună decât configurația S. Spre deosebire de Simulated Annealing care folosește direct valoarea funcției obiectiv pentru calculul probabilității de acceptare a unei noi configurații, în Tabu Search este suficient să se decidă care dintre soluții este mai bună, ceea ce permite tratarea restricțiilor și fără a folosi tehnica penalizării (folosind in schimb reguli de fezabilitate – vezi 3.2.).

3.2. Rezolvarea problemei rucsacului folosind Tabu Search

a) Codificarea soluției: vector binar
b) Perturbare locală: modificarea (prin complementare) a unei componente
c) Structura listei "tabu": soluții candidat (vectori binari)
d) Compararea a două soluții (reguli de fezabilitate):
   - Dacă S și S' sunt fezabile atunci este mai bună varianta cu valoarea mai mare
   - Dacă una dintre soluții este fezabilă iar cealaltă nu este fezabilă atunci soluția fezabilă este considerată mai bună
   - Dacă nici una dintre soluții nu este fezabilă atunci cea care încalcă mai puțin restricția (depășește cu mai puțin capacitatea rucsacului) este considerată mai bună

Aplicație 3. Implementați un algoritm de tip Tabu Search pentru a rezolva problema rucsacului.

Indicație: o variantă de implementare este în TS_Knapsack.sci

Exercițiu: Analizați influența dimensiunii maxime a listei de configurații interzise asupra comportării algoritmului.

Temă.

a) Adaptați implementarea algoritmului Tabu Search pentru a rezolva problema comis-voiajorului
b) (facultativ) Modificați implementarea astfel încât în lista tabu să nu fie reținute soluții candidat ci transformări applicate acestora (vezi Feature-Based TS, [S. Luke, Essential of Metaheuristics, pg 27]

Anexă:

1. **Generarea valorilor aleatoare în SciLab**

Valori aleatoare corespunzătoare diferitelor distribuții de probabilitate pot fi generate folosind funcția grand

Forma general de apel, care returnează o matrice cu valori aleatoare de dimensiune mxn este:

grand(m,n,"<specificator distribuție>", <parametri distribuție>)

Exemple:

1. Valori aleatoare intregi uniform repartizate in mulțimea {inf,inf+1,inf+2,…,sup}

   grand(m, n, "uin", inf, sup)

2. Valori aleatoare reale uniform repartizate în intervalul [inf,sup)

   grand(m, n, "unf", inf, sup)

3. Valori aleatoare reale cu repartiție normal cu media m si abaterea standard s

   grand(m, n, "nor", m, s)

O altă funcție utilă este cea care generează n permutări aleatoare ale vectorului vect:

grand(n, "prm", vect)

2. **Citirea fișierelor text**

   Fișierele de tip csv pot fi citite folosind csvRead. Varianta cea mai simplă de apel este csvRead('nume fisier') (pentru cazul când valorile sunt separate prin virgulă. In cazul în care este folosit alt separator se specific csvRead('nume fisier','separator').

   Există și alte functii de citire din fișiere: read, fscanfMat