Căutare globală folosind un singur candidat

- Tabu Search
- Variable Neighborhood Search
- Guided Local Search
- GRASP = Greedy Randomized Adaptive Search Procedure

Tabu Search

Căutare bazată pe liste de configurații interzise (TS – Tabu Search)

Creator: Fred Glover (1986)

Scop: rezolvarea problemelor de optimizare combinatorială

Specific:

- Tehnică iterativă de căutare locală bazată pe explorarea vecinătății configurației curente
  - vecinătatea unei configurații se definește ca fiind mulțimea configurațiilor ce pot fi atinse din configurația curentă printr-o singură transformare;
  - transformările posibile sunt specifice problemei

Tabu Search

- Exemple de vecinătăți pentru o configurație *S*
  - TSP: configurații noi se pot obține prin aplicarea transformării 2-opt asupra configurației S
  - Problema rucsacului: generarea de vectori binaricare sunt la o distanța Hamming egală cu 1 de S
  - Probleme de asignare (bin paking):
    - Mutarea unui obiect dintr-un bin în altul
    - Interschimbarea poziției a două obiecte care aparțin la bin-uri diferite

- Utilizează o listă de configurații interzise care nu vor putea fi vizitate în următoarele iterații; lista tabu are o dimensiune limitată (e implementată ca listă circulară)

Tabu Search

Căutare bazată pe liste de configurații interzise (TS – Tabu Search) Structura generala:

Pas 1: se construiește o configurație inițiala (*S*)

S* = S // S* cea mai bună configurație de până acum

T = [S] //inițializare listă tabu

Pas 2: REPEAT

- Se selectează cel mai bun element, S', din vecinătatea configurației curente, *N(S)*, care este acceptabil în raport cu lista tabu
- Dacă elementul selectat este suficient de bun atunci se adaugă la o arhivă cu elite (if s(S`)<f(S*) then S*=S'
- S = S'
- Se actualizează lista tabu prin includerea lui S, dacă dimensiunea listei tabu este prea mare se înlătură cel mai vechi element

UNTIL <conditie de oprire>

Obs:

1. Dacă vecinătatea unei configurații este prea mare atunci nu se evaluează toate elementele ci doar o selecție a acestora
2. Pentru eficientizarea lucrului cu lista tabu în loc de configurații se pot stoca caracteristici ale acestora sau descrieri ale transformărilor efectuate

Tabu Search

Pentru eficientizarea lucrului cu lista tabu în loc de configurații se pot stoca caracteristici ale acestora sau descrieri ale transformărilor efectuate

Exemple de caracteristici:

- In cazul unei reprezentări de tip permutare interschimbarea a două elemente poate reprezenta o astfel de caracteristică. De exemplu (*i,j*) specifică faptul că elementul de pe pozitia *i* a fost interschimbat cu elementul de pe pozitia *j*
- In cazul unei reprezentări de tip vector binar complementarea valorii unei componente poate reprezenta o caracteristică. De exemplu (*i,0*) reprezintă faptul că pe poziția *i* este plasată valoarea *0*, iar (*i,1*) faptul că pe poziția *i* este plasată valoarea *1*.

Obs: utilizarea caracteristicilor în loc de liste complete poate conduce la situații în care se restricționează căutarea prin excluderea unor configurații foarte bune. Pentru a evita astfel de situații se permite acceptarea unor configurații bune chiar dacă corespund unor caracteristici trecute în lista tabu.

Tabu Search

Pentru îmbunătățirea funcționării se pot aplica periodic etape de intensificare și diversificare a căutarii

Intensificare:

Scop: exploatarea regiunilor ce par promițătoare (ex. penalizarea configurațiilor care sunt depărtate de cea curentă)

Mod de implementare:

- Se contorizează pentru fiecare componentă a configurației curente numărul de iterații consecutive în care a rămas nemodificată
- Se restartează procesul de căutare pornind de la cea mai bună configurație întâlnită și considerând fixate componentele cu comportare bună (pentru care contorul are valori mari)

Alte variante

Pentru îmbunătățirea funcționării se pot aplica periodic etape de intensificare și diversificare a căutarii

Diversificare:

Scop: explorarea regiunilor ce nu au fost vizitate (ex. penalizarea configurațiilor care sunt apropiate de cea curentă)

Mod de implementare:

- Se contorizează frecvența de utilizare, pe parcursul întregului proces iterativ, a valorilor corespunzătoare diferitelor componente
- Se restartează procesul de căutare de la configurații în care sunt plasate componente cu frecvența mică de utilizare; sau se penalizează scorul unei configurații folosind frecvențele asociate componentelor; relaxarea restricțiilor prin modificarea sistematică (creștere urmată de descreștere și invers) a ponderilor utilizate în funcția obiectiv care include restricțiile (construită prin tehnica penalizării)

Variable Neighborhood Search

Căutare bazată pe vecinătăți variabile (VNS - Variable Neighborhood Search)
[Mladenovic, P. Hansen, 1997]

- Idee: utilizează o structură de vecinătăți V_1, V_2,...,V_{kmax} care este explorată incremental; in cadrul fiecarei vecinătăți căutarea se realizează utilizând o metodă de căutare locală
- Obs: Structura de vecinătăți ale unei configurații (soluții candidat) x se stabilește în funcție de specificul problemei dar astfel încât dacă k1<k2 atunci elementele lui $V_{k1}(S)$ sunt mai „apropiate" de x decât elementele lui $V_{k2}(S)$
- Exemple:
  - pentru problema comis voiajorului $V_k(S)$ poate conține configurațiile obținute din *S* aplicând k interschimbări de noduri (locații) selectate aleator
  - Pentru problema rucsacului $V_k(S)$ poate conține configurațiile obținute din *S* prin modificarea a k componente

Variable Neighborhood Search

Căutare bazată pe vecinătăți variabile (VNS - Variable Neighborhood Search) [Mladenovic, Hansen, 1997]

Structura generală

Inițializează s (aleator în spațiul de căutare)

k=1

WHILE k<=kmax DO

selectează s' aleator din $V_k(s)$;

construiește s'' din s' aplicând o metodă de căutare locală (care nu e obligatoriu să se limiteze la $V_k(s)$)

IF f(s'')<f(s') THEN s=s''; k=1

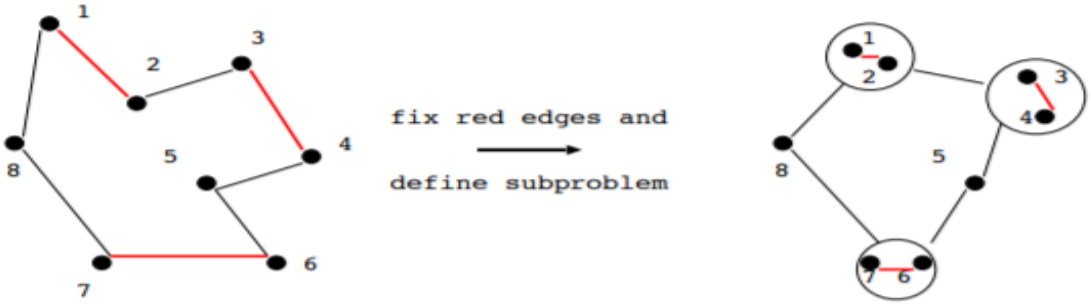
ELSE k=k+1

Observație: Căutrea corespunzătoare iterației k nu este limita $V_k(s)$

Variable Neighborhood Search

Variable Neighborhood Decomposition Search

- Elementul cheie al algoritmilor de tip VNS este alegerea vecinătăților (ceea ce depinde de problema de rezolvat) – acestea controlează structura spațiului de căutare
- O variantă de definire a vecinătăților este cea în care se fixează o parte din atribute (în felul acesta problema inițială se descompune în subprobleme de dimensiune mai mică)
- Numărul de componente fixate este corelat cu dimensiunea vecinătății (de exemplu $V_k(s)$ conține elemente care diferă de s în maxim k componente, celelalte n-k componente fiind fixate). In procesul de căutare locală componentele fixate nu se modifică.



Guided Local Search

Guided Local Search = căutare locală ghidată [Voudouris&Tsang, 1999]

Idee: modificarea dinamică a funcției obiectiv pentru a permite evadarea din minime locale (prin alterarea valorilor asociate funcției obiectiv optimele locale deja vizitate devin mai puțin „dezirabile”)

Modificarea se bazează pe analiza prezenței unor caracteristici în soluția evaluată (de exemplu pt problema comis voiajorului o muchie poate fi considerată drept caracteristică)

$$f_{\text{mod}}(S) = f(S) + \lambda \sum_{i=1}^{m} p_i I_i(S)$$

$$I_i(S) = \begin{cases} 1 & \text{caracteristica } i \text{ este prezenta in } S \\ 0 & \text{caracteristica } i \text{ nu este prezenta in } S \end{cases}$$

$p_i = \text{parametru de penalizare}$

$\lambda = \text{factor de regularizare (controleaza impactul penalizarii)}$

Guided Local Search

Guided Local Search = căutare locală ghidată
[Voudouris&Tsang, 1999]

Structura generală:

$$U(s,i) = I_i(s)\frac{c_i}{1+p_i}$$

c_i = costul caracteristici i

p_i =penalitatea caracteristici i

S=configurație inițială
Repeat
  s=LocalSearch(S,f)
  for <all features i with maximal utility U(s,i)>
    $p_i=p_i+1$
  endfor
  f=Update(f,p)
until <conditie de oprire>

$$f_{\text{mod}}(S) = f(S) + \lambda\sum_{i=1}^{m} p_i I_i(S)$$

$$I_i(S) = \begin{cases} 1 & \text{caracteristica } i \text{ este prezenta in } S \\ 0 & \text{caracteristica } i \text{ nu este prezenta in } S \end{cases}$$

p_i = parametru de penalizare

λ = factor de regularizare (controleaza impactul penalizarii)

GRASP

GRASP = Greedy Randomized Adaptive Search Procedure

Idee: configurația inițială este construită folosind

- O euristică care permite construirea componentă cu componentă a configurației
  - TSP: o locație sau o muchie este adăugată la fiecare pas
  - Problema Rucsacului: la fiecare pas un obiect este selectat
- Alegerea valorii corespunzătoare fiecărei componente se face prin selecție aleatoare dintr-o listă ierarhizată de valori posibile
  - Folosește o selecție neuniforma dintr-o lista de ranguri a elementelor (elementele bune se selectează cu o probabilitate mai mare)
- Dimensiunea listei este variabilă; cazuri extreme:
  - Dimensiunea = 1 -> alegere greedy (se selectează cea mai bună componentă)
  - Dimensiunea = nr maxim de valori posibile pt o componentă -> căutare pur aleatoare

GRASP

```
procedure GRASP(Max_Iterations,Seed)
1     Read_Input();
2     for k = 1,...,Max_Iterations do
3           Solution ← Greedy_Randomized_Construction(Seed);
4           Solution ← Local_Search(Solution);
5           Update_Solution(Solution,Best_Solution);
6     end;
7     return Best_Solution;
end GRASP.
```

```
procedure Greedy_Randomized_Construction(Seed)
1     Solution ← ∅;
2     Evaluate the incremental costs of the candidate elements;
3     while Solution is not a complete solution do
4           Build the restricted candidate list (RCL);
5           Select an element s from the RCL at random;
6           Solution ← Solution ∪ {s};
7           Reevaluate the incremental costs;
8     end;
9     return Solution;
end Greedy_Randomized_Construction.
```


[Resende& Ribeiro, Greedy randomized adaptive search procedures, Handbook in Metaheuristics, 2002]

GRASP

- Pros & cons
  - PRO: Structură simplă (ușor de implementat)
  - PRO: Dacă problema permite rezolvarea folosind un algoritm greedy extensia la GRASP este de obicei simplă
  - PRO: poate fi folosit pentru probleme de optimizare mari
  - CONS: în funcție de algoritmul greedy inițial, se poate bloca în minime locale
  - CONS: poare găsi aceeași soluție de multe ori

Sumar

- Componentele algoritmilor de căutare globală folosind un singur candidat
  - Inițializare
    - Soluția candidat este construită componentă cu componentă (aleator vs greedy)
  - Generarea unui nou candidat
    - Prin construire (folosită de obicei în pasul de inițializare)
    - Prin perturbare – bazata pe selecția (aleatoare vs elitistă) dintr-o vecinătate a configurației curente
      - Vecinătatea poate fi fixă sau variabila (VNS)
      - Terturbarea poate fi mica (cautare locală) sau mare (ILS)
  - Acceptarea unui nou candidat
    - Doar dacă e mai bună decât elementul curent (Hill climbing/descendent)
    - Acceptarea de elemente mai puțin bune (pentru a evita stagnarea sau ciclare)
      - Cu o anumită probabilitate care depinde de calitatea soluției și de parametrii de control (SA)
      - Daca elemente mai bune au fost analizate (TS)

Sumar

- Componentele algoritmilor de căutare globală folosind un singur candidat
  - Condiția de oprire
    - Calitate
      - O configurație de o calitate acceptabilă a fost identificată
    - Comportament
      - Nu se observă îmbunătățiri dea lungul ultimelor iterații
    - Resurse folosite
      - Un număr de repetări sau un număr de evaluări a funcției obiectiv a fost realizat