Hill Climbing (căutare locală simplă)

- alege cel mai bun vecin
- ex: TSP:
  - reprezentarea soluției: o permutare
  - generarea unui vecin: swap 2 random elements
- criterial de acceptare a unei noi soluții: cel mai bun vecin al soluției curente mai bun decât el

Algoritm

```
Bool HC(S){
   x = s1    //starea iniţială
   x*=x      // cea mai bună soluţie găsită
   (până la un moment dat)
   k = 0     // numarul de iteraţii
   while (not termiantion criteria){
     k = k + 1
     generate all neighbours of x (N)
     Choose the best solution s from N
     if f(s) is better than f(x) then x = s
     else stop
   }  //while
   x* = x
   return x*;
}
```

- nu necesită memorie (nu se revine în starea precedentă)
- euristica poate fi dificil de estimat
- aplicații: problema canibalilor, 8-puzzle, 15-puzzle, TSP, problema damelor
- HC stocastic: alegerea aleatoare a unui succesor
- HC cu prima opțiune: generarea aleatoare până la întâlnirea unei mutări neefectuate
- HC cu repornire aleatoare – beam local search: se repornește căutarea la o stare inițială aleatoare atunci când căutarea nu progresează

Simulated Annealing

- succesorii sunt aleși în mod aleator, succesorul "rău" este reținut doar cu o anumită probabilitate
- se evadează din optime locale
- criteriul de acceptare a unei noi soluții: un vecin aleator mai bun sau mai slab (cu probabilitatea p) decât soluția curentă
- aplicații: problema celor 8 regine, problema rucsacului, proiectarea circuitelor digitale, planificarea producției, planificarea meciurilor în turnirurile de tenis US Open
- oprire: s-a ajuns la soluție, s-a parcurs un număr de iterații, s-a ajuns la îngheț
- garantează găsirea soluției optime (global), dar necesită multe iterații
- convergența la soluție durează mult timp

Algoritm

```
bool SA(S){
     x = s1          //starea inițială
     x*=x  // cea mai bună soluție găsită (până la un moment dat)
     k = 0 // numarul de iterații
     while (not termiantion criteria){
         k = k + 1
         generate a neighbour s of x
         if f(s) is better than f(x) then x = s
         else
           pick a random number r (in (0,1) range)
           if r < P(ΔE) then x = s
     }  //while
     x* = x
     return x*;
}
```

Tabu Search

- începe de la o stare oarecare (fără constrângeri)
- se fac modificări îndreptate spre soluția optimă
- se memorează: starea curentă, stările vizitate și mutările efectuate (listă de stări care nu mai trebuie revizitate)
- criteriul de acceptare a unei noi soluții: cel mai bun vecin mai bun decât soluția curentă și nevizitat încă

```
bool TS(S){
    Select x∈S   //S - spaţiul de căutare
    x*=x         //cea mai bună soluţie (până la un mom. dat)
    k = 0        // numarul de iteraţii
    T = ∅        // listă de mutări tabu
    while (not termiantion criteria){
       k = k + 1
       generate a subset of solutions in the neighborhood N-T of x
       choose the best solution s from N-T and set x=s.
       if f(x)<f(x*) then x*=x
       update T with moves of generating x
    }  //while
    return x*;
```

- criterii de terminare: număr fix de iterații, număr de iterații fără îmbunătățiri, apropierea suficientă de soluție (dacă e cunoscută), epuizarea elementelor nevizitate dintr-o vecinătate
- algoritm rapid
- număr mare de iterații, nu garantează atingerea optimului global
- aplicații: optimizarea traficului, probleme în grafuri, planificări ale task-urilor paralele efectuate în procesor, probleme de combinatorică (plata sumei), NASA, probleme de transport

Best first search

- nodurile de cost mai mic au prioritate
- greedy:
    - minimizează costul de la starea curentă la starea finală (h(n))
    - ne-completă și ne-optimală

- de tip A*:
    - minimizează costul de la starea inițială la cea curentă (g(n)) și de la curentă la finală (h(n)) → f(n)
    - evită repetarea stărilor
    - timp și spațiu de căutare mare
    - complet și optimal

SC locale

	Alegerea stării următoare	Criteriul de acceptare	Convergenţa
HC	Cel mai bun vecin	Vecinul este mai bun decât strarea curentă	Optim local sau gloabl
SA	Un vecin oarecare	Vecinul este mai bun sau mai slab (acceptat cu probabilitatea p) decât starea curentă	Optim global (lentă)
TS	Cel mai bun vecin nevizitat încă	Vecinul este mai bun decât strarea curentă	Optim global (rapidă)

Algoritmi evolutivi (strategie de căutare locală)

- procese iterative și paralele
- folosesc populații de potențiale soluții
- bazată pe căutare aleatoare

Evoluţia naturală		Rezolvarea problemelor
Individ	↔	Soluţie potenţială
Populaţie	↔	Mulţime de soluţii potenţiale
Cromozom	↔	Codarea unei soluţii potenţiale
Genă	↔	Parte a codării
Fitness	↔	Calitate
Încrucişare şi mutaţie	↔	Operatori de căutare
Mediu	↔	Problemă

- 2 nivele de existență pentru o soluție candidat:
    - nivel exterior – fenotip
        - individ = obiectul original

        - aici are loc evaluarea unei potențiale soluții
        - ex: furnică, rucsac, elefantul Cici, orașe etc.
    - nivel interior – genotip
        - cromozom – codul asociat unui obiect
        - aici are loc căutarea unei noi potențiale soluții
        - ex: vector unidimensional, matrice etc.

Populație:

- scop: reține o colecție de soluții candidat
- folosită în selecție pentru reproducere
- dimensiune fixă
- unitatea de bază care evoluează

Modele de populații:

- Generațional:
    - în fiecare generație de creează un număr de descendenți
    - individul supraviețuiește o singură generație
    - părinții sunt înlocuiți în întregime
- Steady-state:
    - în fiecare generație se obține un singur descendent
    - cel mai slab părinte e înlocuit cu descendentul

Funcția de evaluare:

- reflectă condițiile la care trebuie să se adapteze populația
- funcție de calitate (fitness)
- asociază o valoare fiecărui candidat

Selecția:

- acordă șanse de reproducere mai mari indivizilor mai buni
- direcționează spre îmbunătățirea calității
- lucrează la nivel de populație
- se bazează doar pe fitness

Operatori de variație:

- generarea unor soluții potențiale noi
- lucrează la nivel de individ
- se bazează pe reprezentarea indivizilor

Mutația:

- scop: reintroducerea în populație a materialului genetic pierdut
- operator UNAR
- reprezentarea binară: schimbarea cu probabilitatea p_m a unor gene în complementul lor (**mutație tare**)

- reprezentarea binară: schimbarea cu probabilitatea p_m a unor gene în 0 sau 1 (**mutație slabă**)
- reprezentarea întreagă: **random resetting**: se schimbă valorile unor gene (cu probabilitatea p_m)
- reprezentarea întreagă: **mutație creep**: valoarea unei gene se schimbă prin adăugarea unei valori (cu probabilitatea p_m) – modificare mică
- reprezentarea permutare: **swap mutation**
- reprezentarea permutare: **mutație prin inserție**: aleg 2 gene și se inserează a doua după prima
- reprezentarea permutare: **mutație prin inversare**
- reprezentarea permutare: **scramble mutation**
- reprezentarea reală: **mutație uniformă**: se schimbă o valoare random cu probabilitatea p_m la o valoare aleasă aleator uniform din domeniu

Crossover:

- amestecarea informațiilor preluate din părinți
- prin n puncte de tăietură

Algoritm generaţional

```
Iniţializare P(0)
Evaluare(P(0))
g = 0;
while (not condiţie_stop) do
    repeat
        Selectarea a 2 părinţi p1 şi p2 din P(g)
        Încrucişare(p1, p2) → o1 şi o2
        Mutaţie(o1) → o1*
        Mutaţie(o2) → o2*
        Evaluare(o1*)
        Evaluare(o2*)

        Adăugare o1* şi o2* în P(g+1)
    until P(g+1) este plină
    g++
endWhile
```

Algoritm steady-state

```
Iniţializare P
Evaluare(P)

while (not condiţie_stop) do
    For i = 1 to |P|
        Selectarea a 2 părinţi p1 şi p2 din P(g)
        Încrucişare(p1, p2) → o1 şi o2
        Mutaţie(o1) → o1*
        Mutaţie(o2) → o2*
        Evaluare(o1*)
        Evaluare(o2*)
        B = Best(o1*, o2*)
        W = Worst(o1*, o2*)
        Dacă B e mai bun ca W, W ← B
    EndFor

endWhile
```

PSO

- căutare cooperativă, ghidată de calitatea relativă a indivizilor
- metodă de optimizare bazată pe populații de particule care cooperează (în loc de competiție ca în cazul GA)
- fiecare particulă se mișcă și are o viteză anume, reține locul unde a obținut cele mai bune rezultate, are asociată o vecinătate de particule
- particulele schimbă informații între ele

- fiecare particulă știe fitness-ul vecinilor, poate folosi poziția celui mai bun vecin pentru a-și ajusta propria viteză
- un individ are memorie

Inițializare:

- fiecare particulă are asociată:
  - poziție – potențială soluție
  - viteză – modifică poziția
  - fitness
- fiecare particulă trebuie să poată:
  - interacționa cu vecinii
  - memora o poziție precedentă
  - utiliza informațiile pentru decizii
- poziții aleatoare, viteze aleatoare sau nule

Actualizarea memoriei:

- stabilirea celei mai bune particule din swarm
- stabilirea celei mai bune poziții

Aplicații: clasificarea cancerului, comunicare în rețele, optimizări financiare, robotică, planificare

ACO

- căutare cooperativă, ghidată de calitatea relativă a indivizilor
- furnicile construiesc soluția plimbându-se prin graf și depunând pe muchii feromoni
- fiecare furnică:
  - se mișcă și depune o cantitate de feromon pe drumul parcurs
  - reține drumul parcurs
  - alege drumul în funcție de: feromonul existent pe drum și informația euristică asociată drumului
  - cooperează cu celelalte furnici prin feromoni care depind de calitatea soluției și care se evaporă cu trecerea timpului
- aplicații: drum optim (TSP), optimizări în rețele, transport

$y = mx + b$

$m = suma[(xi - x)(yi - y)] / sum[(xi - x)^2]$

$y = 37 / 5 = 7.4$

$x = 167 / 5 = 33.4$

$suma[(xi - x)(yi - y)] = (11 – 33.4)(4 – 7.4) + \ldots$