SIMULIRANO KALJENJE (SIMULATED ANNEALING)

- podvrsta stohastičkih optimizacijskih algoritama
- drugi nazivi: *Monte Carlo kaljenje*, *stohastičko hlađenje*
- način rada klasičnog (determinističkog) optimizacijskog postupka:
  - kreće od jednog početnog rješenja
  - postojeće rješenje zamjenjuje boljim, iz neposredne okoline
  - uvijek nalazi najbliži lokalni optimum
- simulurano kaljenje:
  - kreće od jednog početnog rješenja
  - postojeće rješenje zamjenjuje boljim, ali ga može zamijeniti i lošijim, uz određenu vjerojatnost prihvaćanja
  - vjerojatnost prihvaćanja lošijeg rješenja opada kako algoritam napreduje
  - nalazi (i) globalni optimum
- priča o kaljenju metala…
- preslikavanje u algoritamsku domenu:
  - konfiguracija atomske rešetke – moguće rješenje problema
  - energija – funkcija cilja
  - temperatura – kontrolni parametar *c*

```
procedura simulirano_kaljenje(i0,c0);
   i:=i0;     // početno rješenje
   c:=c0;
   Ci:=C(i); // funkcija cilja
   ponavljaj
      ponavljaj
         j:=susjedno_rješenje(i);
         Cj:=C(j);
         ΔC:=Cj-Ci;
         prihvati:=FALSE;
         ako je ΔC<0 tada
              prihvati:=TRUE;
         inače
              ako je exp(-ΔC/c)>random[0,1] tada
                 prihvati:=TRUE;
         ako je prihvati=TRUE tada
              i:=j;  // prihvati susjedno rješenje
              Ci:=Cj;
      do termalne_ravnoteže
      smanji parametar c;
   do zamrzavanja
kraj.
```

- u općenitom slučaju treba odabrati:
  - početnu vrijednost c_0
  - konačnu vrijednost c_F
  - funkciju hlađenja, odnosno kako se mijenja c
- određivanje početnog c

- zadajemo početnu vjerojatnost prihvaćanja koja je relativno velika (>50%) - p_0
- odredimo prosječno pogoršanje (tj. povećanje) funkcije cilja za nekoliko susjednih rješenja - ΔC^+
- c_0 računamo kao: $c_0 = \Delta C^+ / \ln(1/p_0)$

- Konačna vrijednost c_F se najčešće ne zadaje, nego se postupak ponavlja zadani broj puta (vanjska petlja)
- funkcija hlađenja se najčešće realizira množenjem c sa brojem manjim od 1 (iz [0.5,0.99])
- broj ponavljanja unutarnje petlje ("termalna ravnoteža") se obično zadaje kao brojčana vrijednost ovisna o veličini (složenosti) problema

RJEŠAVANJE TSP-A POMOĆU SIMULIRANOG KALJENJA

- TSP (travelling salesman problem) – problem trgovačkog putnika; najpoznatiji problem kombinatoričke optimizacije
- obići N gradova uz najmanji put
- (N-1)! mogućih rješenja (NP težak problem)
- predstavljanje rješenja: niz indeksa gradova (4,7,2,…) duljine N
- kako odrediti "susjedno rješenje"?
  - najjednostavnije: zamjena dva slučajno odabrana grada
  - (4,**7**,2,3,**5**,8) → (4,**5**,2,3,**7**,8)
  - složenije ali učinkovitije (isprobajte!): zamjena redoslijeda podniza gradova
  - (4,**7**,2,3,**5**,8) → (4,**5**,**3**,**2**,**7**,8)
- kako odrediti početni parametar c_0? moramo odrediti prosječno povećanje puta (u nekoliko slučajnih promjena)
- ulazni parametri:
  - N – broj gradova (100)
  - S – broj ponavljanja vanjske petlje (10-100)
  - p_0 – početna vjerojatnost prihvaćanja lošijeg rješenja (0.7-0.8)
  - α – faktor smanjenja 'temperature' (0.5-0.99)
  - KTL – koeficijent termalne ravnoteže: pomoću njega računamo broj ponavljanja unutarnje petlje (0.1-0.5)
- ostale varijable:
  - G – matrica koordinata gradova (dimenzija Nx2)
  - D – matrica udaljenosti gradova (dimenzija NxN)
  - put – rješenje s kojim radimo, niz od N brojeva
  - dput – duljina puta (vrijednost funkcije cilja)
  - promjena – razlika duljina puteva

```
procedura TSP(N,S,p0,α,KTL);
   (slučajno) generiraj koordinate N gradova u kvadratnom području;
   // ili učitaj koordinate iz datoteke, npr.
   izračunaj matricu D(NxN); // udaljenosti svih mogućih kombinacija gradova
   put:=početno rješenje; // (1,2,3,...,N)
   dput:=duljina(put);

   // određivanje prosječnog povećanja puta - funkcije cilja
   prosjek_povecanja:=0; brojac:=0;
   za i:=1 do 100 radi
      put2:=slučajno_susjedno_rješenje(put);
      dput2:=duljina(put2);  // ne računati za sve gradove!
      ako je dput2>dput tada // samo u slučaju povećanja duljine puta
         prosjek_povecanja+=(dput2-dput);
         brojac++;
   c:=(prosjek_povecanja/brojac)/ln(1/p0); // početni c

   za i:=1 do S radi {
      za j:=1 do KTL*N2 radi {
         put2:=slučajno_susjedno_rješenje(put);
         dput2:=duljina(put2);
         promjena:=dput2-dput;
         prihvati:=FALSE;
         ako je promjena<0 tada
              prihvati:=TRUE;
         inače
              ako je exp(-promjena/c)>random[0,1] tada
                 prihvati:=TRUE;
         ako je prihvati=TRUE tada
              put:=put2;
              dput:=dput2;
         } // unutarnja petlja
      c:=c*α;
   } // vanjska petlja
kraj.
```