Algoritmi de aproximare si euristici

- Algoritmi de aproximare pentru probleme de optim
  - ⇨Definitii
  - ⇨Studii de caz:
    - acoperirea unei multimi
    - submultime de suma data
    - comis voiajor
- Euristici
  - ⇨Un prim exemplu: cautare locala
  - ⇨Definitie
  - ⇨Alte exemple
    - calire simulata (simulated anealing)
    - cautare tabu

Ce e defacut cind avem de rezolvat o problema NP-dificila?

1. Renunta la algoritmii polinomiali si spera ca algoritmul proiectat rezolva in timp rezonabil problema pentru instantele intalnite in practica (backtracking, branch-and-bound – cursul trecut).
2. Renunta la optimalitate si incearca una din urmatoarele posibilitati:
   a) algoritmi de aproximare
   b) euristici
      – cautare locala
      – cautare tabu
      – "hill climbing" (cursul de IA)
      – "simulated anealing"
   c) algoritmi genetici (optional anul II)

Algoritmi de aproximare

- Este fascinant de vazut cum uneori o schimbare minora a cerintelor duce la un salt de la complexitate exponentiala (netractabil) la complexitate polinomiala (tractabil).
- Aceasta schimbare poate fi de forma: in loc de solutia optima, se cere o solutie care difera de solutia optima cu cel mult $\varepsilon\%$, $\varepsilon > 0$.
- Abordarea pare una potrivita pentru algoritmii cu aplicabilitate practica mare.
- Principalele probleme la care trebuie sa raspundem:
  - ⇨ **fundamentele conceptului de aproximare**
  - ⇨ **exemple**
  - ⇨ **margini inferioare pentru neaproximabilitate polinomiala in timp**

Algoritmi de aproximare: fundamente

- Fie P = problema de optim $\mathcal{NP}$-completa.
- Pentru o intrare x cu dimensiunea g(x) notam:
  - ⇨ Optim(x) = valoarea optima
  - ⇨ A(x) = valoarea calculata de A

Definim:

- Un **algoritm de aproximare** pentru P este un algoritm A care determina o solutie aproape optima.
- **Eroarea relativa** a alagoritmului A este definita prin:

$$E_A(x) = \frac{|A(x) - Optim(x)|}{Optim(x)}$$

$$E_A(n) = \max\left\{ \frac{|A(x) - Optim(x)|}{Optim(x)} \middle| g(x) = n \right\}$$

Algoritmi de aproximare: fundamente

- A este un algoritm de aproximare cu **eroarea relativa marginita de ε(n)** daca**:**

$$E_A(n) \leq \varepsilon(n)$$

- **Ratia de aproximare** a lui A este definita prin:

$$R_A(x) = \max\left(\frac{A(x)}{Optim(x)}, \frac{Optim(x)}{A(x)} \right)$$

$$R_A(n) = \max\{R_A(x) \mid g(x) = n\}$$

- A este algoritm de aproximare cu **ratia marginita de ρ(n)** daca

$$R_A(n) \leq \rho(n)$$

- Daca stim ratia putem aproxima eroarea: **ε(n) ≤ ρ(n)** – 1.

Exemplu: Acoperirea unei multimi

- Intrare:
  - ⇨o multime $T = \{t_1, t_2, ..., t_n\}$
  - ⇨m submultimi: $S_1, S_2, ..., S_m \subseteq T$

    de ponderi (costuri) $w_1, w_2, ..., w_m$
- Iesire: o submultime $I \subseteq \{1,2,...,m\}$ care minimizeaza $\sum_{i \in I} w_i$ astfel incat $\bigcup_{i \in I} S_i = T$

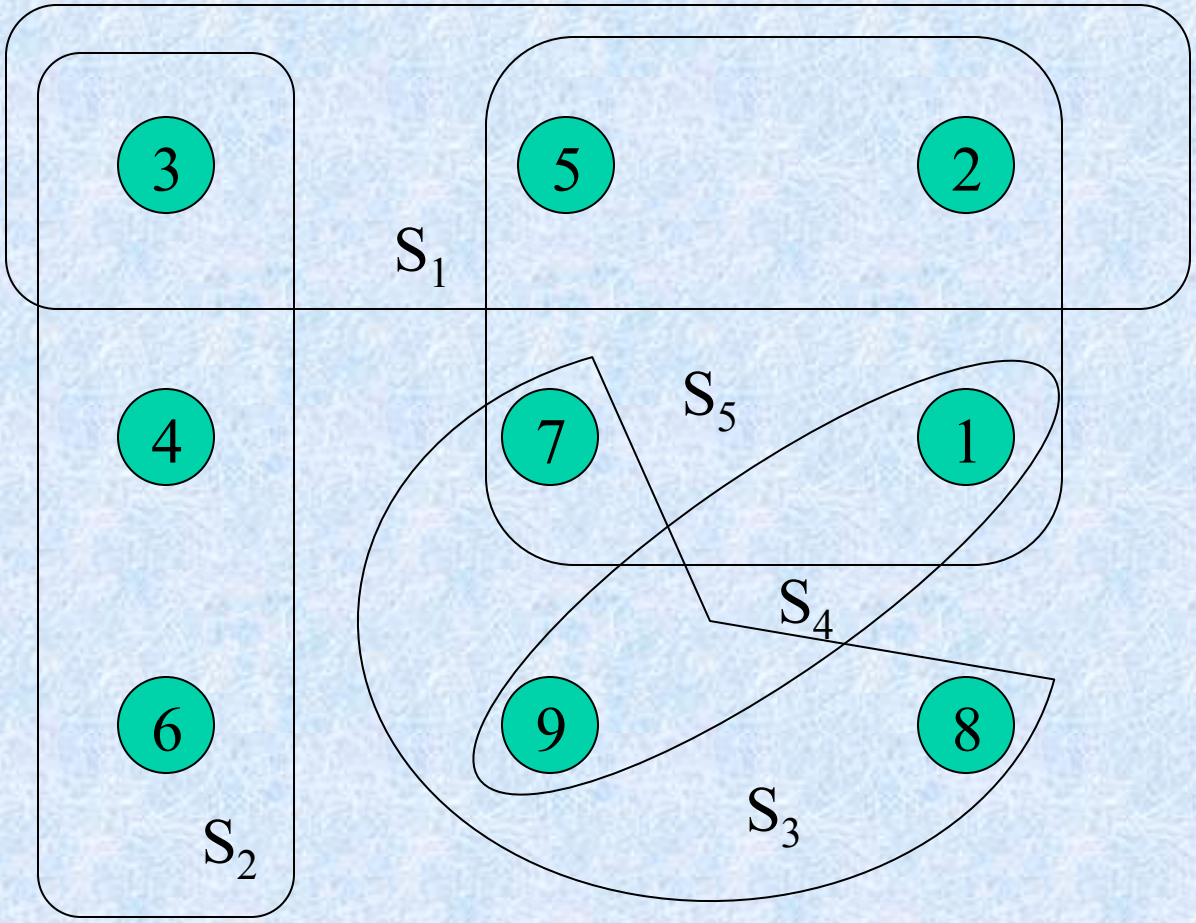
- V-acoperire este un caz special
  - ⇨T = multimea muchiilor
  - ⇨S_i = multimea muchiilor incidente in i
  - ⇨$w_i = 1$

Exemplu: Acoperirea unei multimi

- Algoritm "greedy" (versiunea 1)

```
asGreedy1(T, S, w)
{
   I = Ø;
   while (T ≠ Ø) {
      (*)alege t_i din T;
      I = I ∪ {j | t_i ∈ S_j};
      T = T - ∪_{j∈I} s_j;
   }
}
```

Exemplu: Acoperirea unei multimi



$t_1 = 3$
$I = \{1, 2\}$

$t_2 = 7$
$I = \{1, 2, 3, 5\}$

Exemplu: Acoperirea unei multimi

Teorema

Daca toate ponderile sunt 1 (cazul neponderat), atunci **`asGreedy1`** este un algoritm de aproximare cu ratia marginita de ρ, unde $\rho = \max_i |\{j \mid t_i \in S_j\}|$

Demonstratie

- ⇨ Fie IO(T,S,w) cu *Optim*(T,S,w) = |IO(T,S,w)|.
- ⇨ Pp. ca exista $t_i \neq t_j$ alesi in (*) a.i. t_i , $t_j \in S_k$ pentru un k oarecare din solutia optima IO(T,S,w)
  - pp. ca t_j s-a ales dupa t_i
  - cind a fost ales t_i, S_k a fost eliminata din T
  - deci t_j nu poate fi ales mai tarziu; contradictie!
- ⇨ Rezulta ca fiecare t_i ales de (*) apartine la un element distinct din solutia IO(T,S,w).

Exemplu: Acoperirea unei multimi

- ⇨ Fie X numarul de iteratii ale buclei while.
- ⇨ Avem $X \leq |IO(T,S,w)|$ (din observatia din slide-ul precedent).
- ⇨ Rezulta $|I| \leq \rho \cdot X \leq \rho \cdot |IO(T,S,w)|$.
- ⇨ Deoarece *asGreedy1*(T,S,w) = |I|,

  obtinem *asGreedy1*(T,S,w) / *Optim*(T,S,w) $\leq \rho$

Exemplu: Acoperirea unei multimi

- Algoritm "greedy" (versiunea 2)
  - ⇨ Primul algoritm nu tine cont de ponderi, asa ca putem imbunatati criteriul de alegere locala tinand cont de ponderi.
  - ⇨ Fie C = multimea elementelor deja acoperite (= $\{t_i \mid i \in I\}$).
  - ⇨ Definim **cost efectiv** al lui S_j ca fiind $w_j / |(S_j - C)|$

```
asGreedy2(T, S, w) {
  I = C = Ø;
  while (T ≠ C) {
    determina Sj cu cel mai mic cost efectiv;
    foreach (ti ∈ Sj-C)
       pret(ti) = wj/|(Sj - C)|;
    I = I ∪ {j};
    C = C ∪ Sj;
}
```

Exemplu: Acoperirea unei multimi

- **Lema**

  Are loc *pret*(t_k) ≤ *Optim*(T,S,w) / (n-k+1).

  Dem.

  ⇨ Se presupune ca elementele din T sunt numerotate in ordinea in care sunt acoperite.

  ⇨ La fiecare iteratie exista cel putin o multime cu costul efectiv ≤ *Optim*(T,S,w) / |(T-C)|.

  ⇨ La momentul in care t_k este acoperit sunt cel putin (n-k+1) elemente nealese (care se gasesc in T – C), care implica

  pret(t_k) ≤ *Optim*(T,S,w) / |(T – C)| ≤ *Optim*(T,S,w) /(n-k+1)

- **Teorema**

  `asGreedy2` este un algoritm de aproximare cu ratia marginita de $H_n = 1 + 1/2 + \ldots + 1/n$.

Scheme de aproximare: fundamente (cont.)

- **Schema de aproximare** = un algoritm de apoximare A pentru P care are la intrare o atat o instanta a lui P cat si un $\varepsilon > 0$ a.i., pentru orice ε fixat, eroarea relativa a lui A este marginita de ε.

- **Schema de aproximare polinomiala** = schema de aproximare care pentru un ε fixat, are complexitatea timp polinomiala in marimea instantei n

- **Schema de aproximare polinomiala completa** = schema de aproximare cu eroarea relativa marginita de ε si care are complexitatea timp polinomiala atit in marimea instantei n cit si in $1/\varepsilon$.

Submultimea de suma data: reformulare

- Intrare: A, $s[a] \in Z_+$, $M \in Z_+$
- Iesire: cel mai mare $M^* \leq M$ a.i. exista $A' \subseteq A$ cu
  $\sum(s[a] \mid a \in A') = M^*$
- Presupunem $A = \{1, 2, ..., n\}$

Submultimea de suma data: algoritm exponential

```
procedure ssdOpt(n, s, M)
{
   L[0] = (0);
   for (i = 1; i <= n; ++i) {
      L[i] = merge(L[i-1], L[i-1] + s[i]);
      elimina din L[i] valorile mai mari decit M;
   }
   return cea mai mare valoare din L[n];
}
```

- L[i] este lista ordonata crescator
- **merge**(L, L') interclaseaza listele L si L'
- $(y_1, \ldots, y_k) + x = (y_1 + x, \ldots, y_k + x)$

Submultimea de suma data: curatirea listei

- Complexitatea exponentiala este data de dimensiunea listelor.
- Asa ca putem obtine algoritmi polinomiali numai daca micsoram dimensiunile listelor.
- Definim:

  ⇨Fie δ cu proprietatea $0 < \delta < 1$. Spunem ca δ **curata** L daca, notand cu L' lista curatata, atunci

  $$(\forall y \in L - L')(\exists z \in L')\ z \leq y \text{ si } (y - z)/y \leq \delta$$

- Exemplu:

  L = (1, 4, 5, 7, 8, 11, 13, 16)

  $\delta = 0.5$ ⇨ L' = (1, 4, 11)

  $\delta = 0.25$ ⇨ L' = (1, 4, 7, 11, 16)

Submultimea de suma data: algoritm de curatire

```
//@input: L lista liniara ordonata crescator
//@output: L' = lista L curatata de δ
curata(L, δ, L') {
   m = lung(L);
   y = L.select(1);
   L' = (y);  // lista cu un singur element
   ultim = y;
   for (i = 2; i <= m; ++i) {
      y = L.select(i);
      if (ultim < (1 - δ)*y) {
        L'.insert(y);
        ultim = y;
      }
   }
}
```

Submultimea de suma data: algoritm de aproximare

```
ssdAprox(n, s, M, ε)
{
   L[0] = (0);
   for (i = 1; i <= n; ++i) {
      Ltemp = merge(L[i-1], L[i-1] + s[i]);
      curata(Ltemp, ε/n, L[i]);
      elimina din L[i] valorile mai mari decit M;
   }
   return cea mai mare valoare din L[n];
}
```

Submultimea de suma data: evaluare

Lema

Fie P[i] multimea valorilor obtinute ca sume de elemente din {s[1], ...,s[i]}.

Pentru orice y ∈ P[i] exista un z ∈ L[i] a.i.

$$\left(1-\frac{\varepsilon}{n}\right)^{i} \cdot y \le z \le y$$

Teorema

Algoritmul ssdAprox este o schema de aproximare complet polinomiala.

Demonstratie

Se considera in lema: i = n, y = y* (solutia optima), z cel dat de algoritmul de aproximare

Submultimea de suma data: evaluare (cont.)

- Se obtine:
  - ⇨ $(1 - \varepsilon)\, y^* \leq z$ (eroarea relativa este marginita de ε)
  - ⇨ $(1 - \varepsilon/n)^n$ este crescatoare
  - ⇨ $(1 - \varepsilon) \leq (1 - \varepsilon/n)^n$
  - ⇨ raportul dintre doua valori consecutive $z/z' > 1/(1 - \varepsilon/n)$
  - ⇨ $M > z_k/z_1 > 1/(1 - \varepsilon/n)^k$
  - ⇨ numarul de elemente din L[n] este cel mult:

$$\log_{\frac{1}{1-\frac{\varepsilon}{n}}} M = \frac{\ln M}{-\ln(1-\frac{\varepsilon}{n})} \leq \frac{n \ln M}{\varepsilon}$$

Submultimea de suma data: exemplu

s = (1001, 1002, 1003, 1004, 1005, 1006, 1007, 1008, 1009, 1010)

M = 4500, ε = 0.5

Ltemp[1] = (0, 1001)

L[1] = (0, 1001)

Ltemp[2] = (0, 1001, 1002, 2003)

L[2] = (0, 1001, 2003)

Ltemp[3] = (0, 1001, 1003, 2003, 2004, 3006)

L[3] = (0, 1001, 2003, 3006)

Ltemp[4] = (0, 1001, 1004, 2003, 2005, 3006, 3007, 4010)

L[4] = (0, 1001, 2003, 3006, 4010)

Ltemp[5] = (0, 1001, 1005, 2003, 2006, 3006, 3008, 4010, 4011, 5015)

L[5] = (0, 1001, 2003, 3006, 4010)

Submultimea de suma data: exemplu (cont. 1)

Ltemp[6] = (0, 1001, 1006, 2003, 2007, 3006, 3009, 4010, 4012, 5016)
L[6] = (0, 1001, 2003, 3006, 4010)

Ltemp[7] = (0, 1001, 1007, 2003, 2008, 3006, 3010, 4010, 4013, 5017)
L[7] = (0, 1001, 2003, 3006, 4010)

Ltemp[8] = (0, 1001, 1008, 2003, 2009, 3006, 3011, 4010, 4014, 5018)
L[8] = (0, 1001, 2003, 3006, 4010)

Ltemp[9] = (0, 1001, 1009, 2003, 2010, 3006, 3012, 4010, 4015, 5019)
L[9] = (0, 1001, 2003, 3006, 4010)

Ltemp[10] = (0, 1001, 1010, 2003, 2011, 3006, 3013, 4010, 4016, 5020)
L[10] = (0, 1001, 2003, 3006, **4010**)

Submultimea de suma data: exemplu (cont. 2)

- Solutia exacta:

0, 1001, 1002, 1003, 1004, 1005, 1006, 1007, 1008, 1009, 1010, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014, 2015, 2016, 2017, 2018, 2019, 3006, 3007, 3008, 3009, 3010, 3011, 3012, 3013, 3014, 3015, 3016, 3017, 3018, 3019, 3020, 3021, 3022, 3023, 3024, 3025, 3026, 3027, 4010, 4011, 4012, 4013, 4014, 4015, 4016, 4017, 4018, 4019, 4020, 4021, 4022, 4023, 4024, 4025, 4026, 4027, 4028, 4029, 4030, 4031, 4032, 4033, **4034**

- eroarea relativa

(4034-4010)/4034 = 0.0053

Comis voiajor - reformulare

- intrare: un graf ponderat $G = (V, E, c)$, $c(\{i,j\}) \in Z_+$
- iesire: un circuit Hamiltonian de cost minim

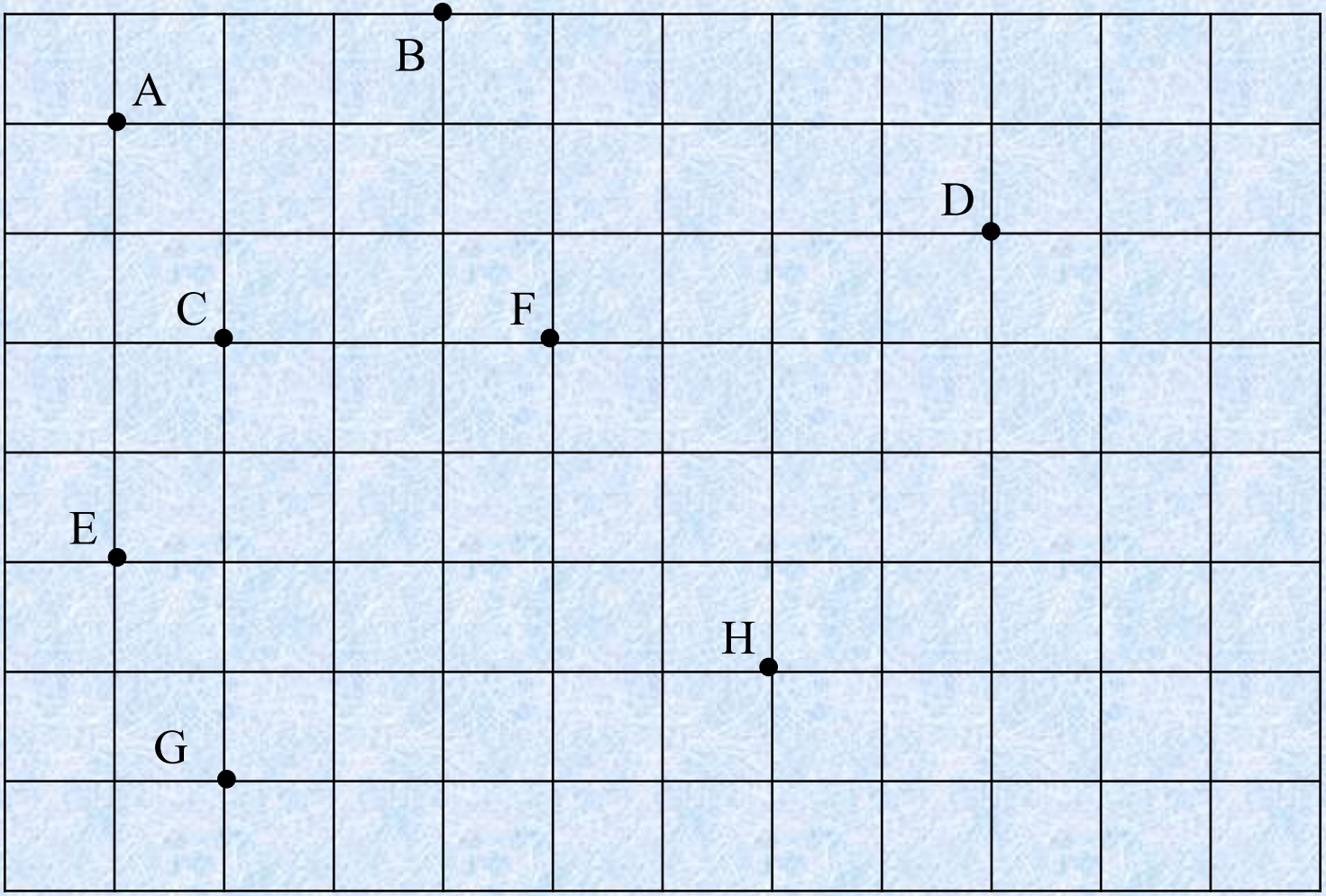
- restrictie: inegalitatea triunghiului

  $(\forall i,j,k)\ c(\{i,j\}) + c(\{j,k\}) \geq c(\{i,k\})$
- problema restrictionata este $\mathcal{NP}$-completa

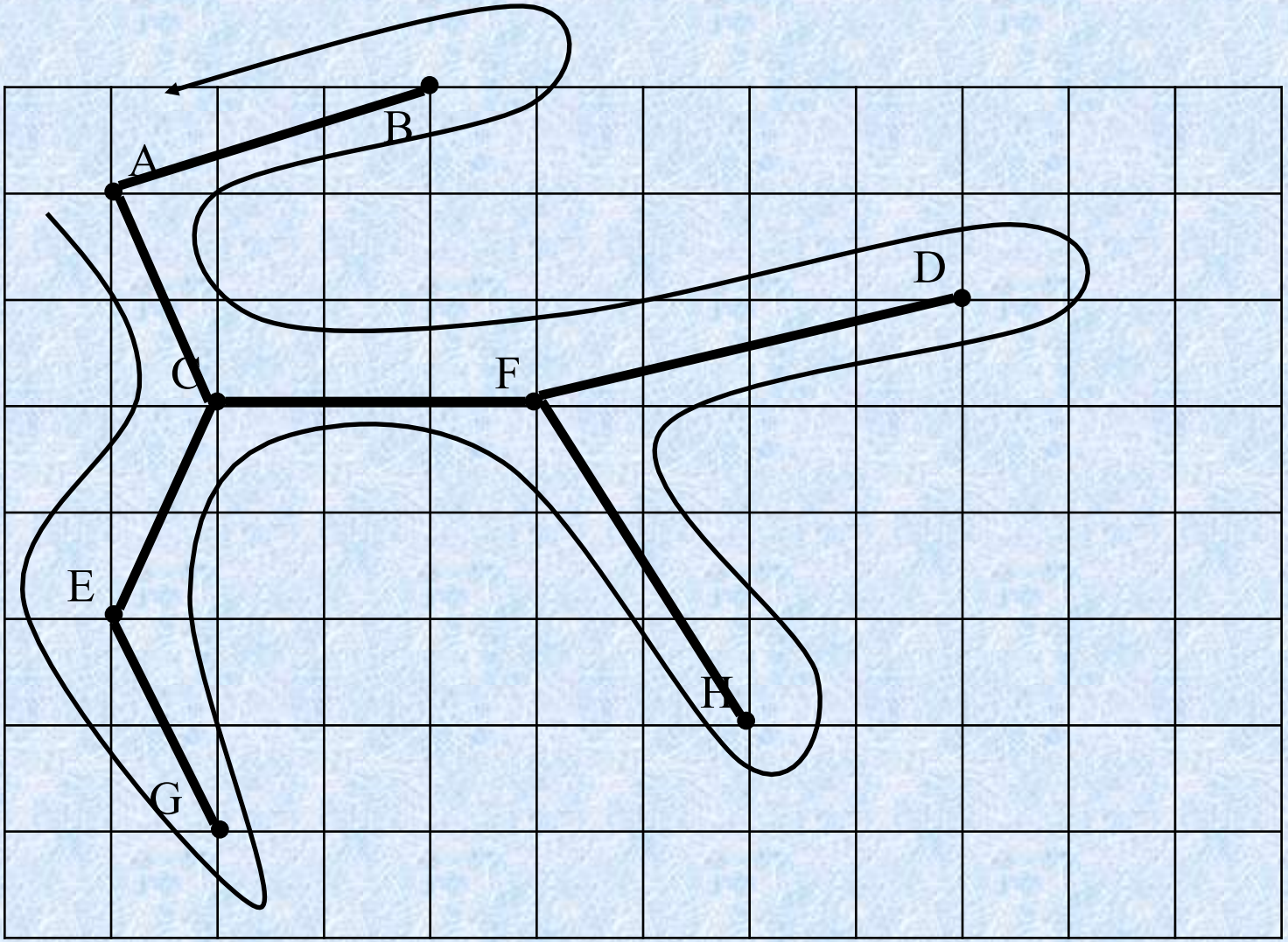
Comis voiajor restrictionata (CVR) - algoritm de aproximare

```
procedure CVRaprox(G, c)
begin
    selecteaza (arbitrar) r din V ca radacina
    determina APCM  T cu radacina r cu algoritmul lui Prim
     fie L lista varfurilor lui T parcurse in preordine
    return L
end
```

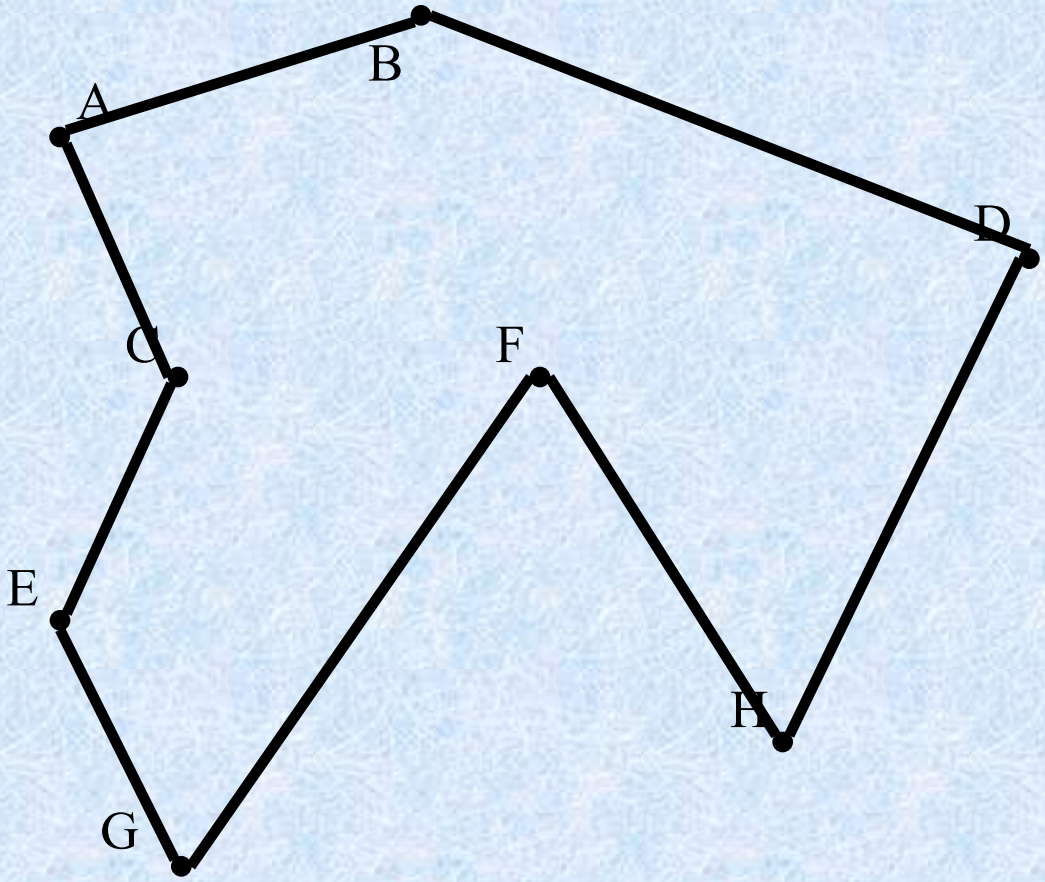
CVR - exemplu - graful



CVR - exemplu - APCM



CVR - exemplu - solutia aproximativa



CV - rezultate

- **Teorema**

  `CVRaprox` este un algoritm de aproximare pentru CVR cu ratia marginita de 2.

- **Teorema**

  Daca $\mathcal{P} \neq \mathcal{NP}$ si $\rho \geq 1$, atunci nu exista un algoritm de aproximare cu ratia marginita de ρ pentru CV general.

Euristici

- Un prim exemplu: cautarea locala
- definitie
- Alte exemple
  - ⇨“simulated annealing’
  - ⇨cautare tabu

Cautare locala: motivatie

- Am vazut ca backtracking si branch-and-bound cauta sistematic in psatiul solutiilor candidat (feasable solutions).
- Din pacate, de multe ori spatiul este foarte mare, sau chiar infinit, si cele doua paradigme nu pot oferi solutii acceptabile.
- Vom discuta metode care nu cauta in tot spatiul, ci numai in anumite zone in ideea de a gasi in medie repede o solutie.
- Aceste metode nu garanteaza gasirea unei solutii chiar daca aceasta exista. De aceea se prefera aplicarea lor atunci cand se stie aprioric ca solutia ca exista sau ca sunt sanse mari sa existe (si astfel cautarea sa se termine cu succes)

Cautare locala: context

- problema P
- o intrare (instanta) x
- U(x) = spatiul solutiilor candidat
- doua solutii candidat α si β sunt **vecine** daca α se obtine din β (si reciproc) printr-un set de transformari locale facute in specificarile lor
- **vecinatatea** unei solutii = multimea solutiilor candidat vecine
- se defineste *merit*(α)
- β **imbunatateste** α daca *merit*(β) > *merit*(α) (are un merit mai mare)
  - ⇨pentru probleme de minim: *cost*(β) < *cost*(α)
  - ⇨pentru probleme de maxim: *profit*(β) > *profit*(α))

Cautare locala: algoritm

- Algoritmul de **cautare locala (iterativa)** este descris de urmatorii pasi:

1. Se alege (arbitrar) o solutie candidat si se calculeaza *meritul* acestei solutii. Se defineste aceasta ca fiind *solutia curenta.*
2. Se aplica o transformare (locala) solutiei curente si se calculeaza meritul noii solutii.
3. Daca meritul noii solutii este mai bun, atunci aceasta devine solutia curenta; altfel noua solutie este ignorata.
4. Se repeta pasii 2 si 3 pana cand nicio transformare nu mai imbunatateste solutia curenta.

Cautare locala si SAT

```
procedure GSAT(F, MAX-TRIALS, MAX-FLIPS)
   for i ← 1 to MAX-TRIALS do
      T ← o atribuire generata aleatoriu
      for j ← 1 to MAX-FLIPS do
         if T satisface formula F
            then return T
            else realizeaza un 'flip' pe baza fct. de merit
      return 'Nu a gasit atribuire care satisface F'
   end
```

- F = formula
- MAX-TRIALS = numarul maxim de secvente de cautare (incercari)
- MAX-FLIPS = numarul maxim de transformari locale

Cautare locala: consideratii generala

- Rareori suntem norocosi sa gasim solutia numai prin incercari.
- Putem defini *meritul* unei solutii candidat astfel incat sa conduca la descresterea numarului de clauze nesatisfacute.
- Aceasta nu garanteaza gasirea unei solutii deoarece cel mai bun flip ar putea conduce de fapt la marirea numarului de clauze nesatisfacute
- Selectia se poate face numai din vecinatatea solutiei; daca toti vecinii (aflati la un flip distanta) nu au merite mai bune, se alege unul cel mai putin “nemerituos”.
- Pentru probleme de optim meritul ar putea fi data de valoarea functiei de optimizat in vecinatate.
- Pericolul este de a nimeri peste un optim local. Algoritmul de cautare locala are sanse sa evadeze dintr-un optim local (dar nu exista garantii: poate oscila la nesfarsit pe un platou).

Euristici 1/2

- Euristica: term ambiguu, utilizat cu diverse intelesuri.
- In sensul general, o euristica pentru o problema de optimizare este un algoritm consistent bazat pe strategie (idee) transparenta de cautare in spatiul solutiilor candidat (feasible solutions) si care nu garanteaza gasirea solutiei optime (aceasta definitie include si algoritmii de aproximare).
- Intr-un sens mai strans, o euristica este o tehnica care furnizeaza (produce) un algoritm pentru care nimeni nu poate dovedi ca acest algoritm gaseste solutii rezonabile intr-un timp rezonabil, dar ideea euristicii promite o comportare buna peste instantele tipice ale problemei de optimizare considerate.

Euristici 2/2

- Din acest p.d.v. algoritmii de aproximare polinomiali nu pot fi considerati euristici.
- O alta proprietate generala a euristicilor este ca sunt *robuste* (o tehnica euristica poate fi aplicata unei clase largi de probleme, chiar cu structuri combinatoriale diferite).
- (Hromkovic)

A heuristic is a robust technique for the design of randomized algorithms for optimization problems, and it provides randomized algorithms for which one is not able to guarantee at once the efficiency and the quality of the computed feasible solutions, even not with any bounded constant probability $p > 0$.

- Daca se elimina termenul “randomized” din definitia de mai sus, atunci cautarea locala poate fi vazuta ca euristica.

“Simulated annealing” (calire simulata)

- O maniera in care sunt formate cristalele

- Se bazeaza pe racirea graduala a lichidului
  - ⇨ la temeperaturi inalte moleculele se misca liber
  - ⇨ la temeperaturi jose ele se “blocheaza”

- Daca racirea este lenta,
  - ⇨ atunci moleculele pot fi structurate in forme regulate (latici), asemnatoare cristalelor

“Simulated annealing” – intuitia pentru euristica

- Se incorporeaza un parametru *temeparatura* in procedura de minimizare
- La temperaturi inalte se exploreaza spatiul
- La temperaturi joase se restructioneaza cautarea
- Cu o “racire” graduala a parametrului de temeperatura, sunt sanse de a da peste solutie
- Metoda a fost descrisa independent de Scott Kirkpatrick, C. Daniel Gelatt si Mario P. Vecchi in 1983, si de Vlado Černý in 1985

“Simulated annealing”

```
simulatedAnnealing {
   x = o valoare initiala de start din S;
   repeat
      x = improve?(x, T);
      update(T);
   until (conditia de terminare)
   return x;
}
```

- Procedura **`improve?(x, T)`** nu intoarce cel mai bun din vecinatate, ci o solutie acceptata din vecinatatea lui x (nu neaparat cea mai buna).
- Parametrul T (*temperatura sintetica*) influenteaza comportarea procedurii **`improve`** si este actualizat permanent.

"Simulated annealing" rafinat

```
procedure SimulatedAnnealing
   t = 0;
   initializeaza T;
   selecteaza aleatoriu val. curenta v_c;
   repeat
       repeat
          selecteaza o val. noua v_n in vecinatatea lui v_c;
          if (eval(v_c) < eval(v_n))
              v_c = v_n;
          else if (random([0,1)) < e^((eval(v_n) - eval(v_c))/T))
              v_c = v_n;
       until (conditie de terminare);
       T = g(T, t);  // g functie de scadere a temperaturii
       t = t + 1;
  until (criteriu de oprire);
end
```

Cautare tabu

- Cuvantul “tabu” vine din Tongan o limba utilizata in Polinezia (o zona din pacific cu peste 1000 de insule) si este utilizat de arborigeni pentru a indica lucruri ce nu pot fi atinse deoarece sunt sacre.
- Abordarea generela consta in a interzice sau penaliza mutari care are conduce la solutii analizate in iteratia urmatoare ce se afla in spatiul deja vizitat (de aici denumirea de tabu).
- Pentru a evita refacerea unor pasi facuti deja, metoda utilizeaza unai sau mai multe liste tabu. Intentia originala nu este neaparat de a nu mai revizita, ci de a vita pasii inapoi.
- metoda oarecum noua, propusa prin 1977

Cautarea tabu

```
cautareTabu
   x =  o valoare initiala de start din S;
   while (!conditia de terminare)  {
      x = improve?(x, H);
      update(H,x);
   return x;
}
```

- **improve?(x, H)** returneaza o solutie acceptata din vecinatatea lui x (nu neaparat cea mai buna), dar bazata acum pe istoria cautarii H
- se utilizeaza o "memorie tabu" H care influnteaza explorarea spatiului de cautare
- se memoreaza cateva solutii tabu (interzise) care vor fi evitate la luarea deciziei de selectare a solutiei urmatoare
- exemplu SAT: pentru ficarea i se memoreaza ultima iteratie la care a fost schimbat ("flipped")

Mai mult la …

- licenta: cursurile de Algoritmi genetici (euristici si metaeuristici), Algoritmica grafurilor (euristici aplicate pentru probleme de combinatorica si grafuri), Inteligenta artificiala (e.g. “hill climbing problem”), Algoritmi de invatare (retele neuronale)
- masterul de optimizare combinatorie