METODE INTELIGENTE DE REZOLVARE A PROBLEMELOR REALE

Laura Dioşan

Tema 2

Conţinut

- Probleme de optimizare combinatorială
  - Problema rucsacului şi problema comisului voiajor
    - Formularea problemei şi exemple
    - Algoritmi de rezolvare
      - Exacţi
      - Euristici
      - Inspiraţi de natură

- De citit:
  - S.J. Russell, P. Norvig – Artificial Intelligence - A Modern Approach → capitolul 3 şi 4
  - H.F. Pop, G. Şerban – Inteligenţă artificială → capitolul 3
  - Documentele din directorul KP şi TSP

Probleme de optimizare combinatorială (POC)

- Definire
  - O problemă P de optimizare (minimizare sau maximizare) care presupune
    - un set de instanţe D_P
    - un set finit de soluţii candidat $S_p(I)$ pentru fiecare instanţă $I \in D_P$
    - o funcţie m_p care asignează unei instanţe I şi unei soluţii candidat $x \in S_P(I)$ un număr raţional pozitiv $m_P(I,x)$, numit valoarea soluţiei
  - Soluţia optimă pentru o instanţă $I \in D_P$ este o soluţie candidat $x^* \in S_P(I)$ a.î. $m_P(I,x^*)$ este mai bună decât $m_P(I,x)$ pentru orice $x \in S_P(I)$

Probleme de optimizare combinatorială (POC)

- Exemple
  - Problema comisului voiajor (Travelling Salesman Problem – TSP)
  - Problema rucsacului
  - Partiţionări în grafe
  - Probleme de atribuiri quadratice
  - Vehicle routing
  - Scheduling

Probleme de optimizare combinatorială (POC)

- Metode de rezolvare
  - Exacte
    - Branch and bound
    - Branch and cut
  - Euristice

- Clasificare
  - Probleme de atribuiri
  - Probleme de aranjare
  - Probleme de partiţionare
  - Probleme de alegere a unor submulţimi

Problema comisului voiajor

- Formularea problemei şi exemple
- Tipologie
- Algoritmi de rezolvare
- Complexităţi

Problema comisului voiajor
Formularea problemei şi exemple

- Se dă
  - un graf (neorientat) (complet), în care cele *n* vârfuri (V) sunt considerate oraşe, iar muchiile (E) drumuri între oraşe (fiecare muchie are asociat un cost).
- Să se găsească
  - cel mai scurt drum care vizitează o singură dată toate oraşele şi se întoarce în oraşul de start
    - ➔ ciclu Hamiltonian

- Dificultate
  - NP-dificilă

- Interes:
  - Problemă de referinţă pentru testarea ideilor

- Denumiri
  - Travelling Salesman Problem (TSP)
  - Canadian Traveller Problem
  - Vehicle routing problem
  - Route inspection problem

Problema comisului voiajor – De ce?

- Problemă conceptual simplă
- Problemă dificil de rezolvat
- Problemă intens cercetată
- Problemă care apare în diverse aplicaţii

Problema comisului voiajor
Instanţe de referinţă

- Consideraţii generale
  - Metrica frecvent folosită – distanţa Euclideană
  - Distanţele - numere întregi

- Instanţe
  - TSPLIB → http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/
    - Peste 100 instanţe cu până la 85900 de oraşe
    - Anumite instanţe sunt preluate din aplicaţii practice
  - Instanţe din proiectarea circuitelor VLSI (Very Large Scale Integration)
    - Împachetarea a cât mai multe dispozitive logice pe suprafeţe cât mai mici
  - Instanţe generate aleator (grupate şi uniforme)
    - 8th DIMACS challenge → http://www2.research.att.com/~dsj/chtsp/

Problema comisului voiajor – Tipologie

- Tipul grafului
  - problema simetrică
    - graf neorientat ➔ nr soluţiilor se înjumătăţeşte
  - problema asimetrică
    - graf orientat
      - coliziuni în trafic
      - străzi cu sens unic

- Tipul distanţelor între 2 noduri
  - metrică → inegalitatea triunghiului $c_{ij} < c_{ik} + c_{kj}$
    - distanţă Euclidiană
    - distanţă Manhattan
  - non-metrică
    - Ex. Traficul aerian

Problema comisului voiajor – Algoritmi

- Exacţi
  - Forţa brută
  - Programare dinamică
  - Branch-and-bound
  - Programare liniară

- Euristici
  - Constructive
  - De îmbunătăţire

Problema comisului voiajor – Algoritmi

- Exacţi
  - **Forţa brută**
  - Programare dinamică
  - Branch-and-bound
  - Programare liniară

- Euristici
  - Constructive
  - De îmbunătăţire

Forţa brută

- Alte denumiri
  - Căutare exhaustivă
  - Generează şi testează

- Mod de lucru
  1. Se generează o potenţială soluţie
  2. Se evaluează această potenţială soluţie
  3. Se verifică dacă costul curent este mai bun decât costul precedent
     - Dacă da, se reţine această soluţie
  4. Se revine la pasul 1

Forța brută – TSP

→ alegerea permutării optime

Algoritm

1. Se generează o permutare
2. Se dermină costul asociat ei
3. Se verifică dacă costul curent este mai bun decât costul precedent
   - Dacă da, se reţine această soluţie
4. Se revine la pasul 1

Problema comisului voiajor – Algoritmi

- Exacţi
  - Forţa brută → alegerea permutării optime
  - **Programare dinamică**
  - Branch-and-bound
  - Programare liniară
- Euristici
  - Constructive
  - De îmbunătăţire

Programare dinamică

- Principii
  - **Principiul optimalităţii**
    - Optimul general implică optimele parţiale
    - Optimul parţial nu implică optimul general

- Mod de lucru
  - Descompunerea problemei în subprobleme
    - Se rezolvă mai întâi subproblemele care pot fi soluţionate imediat
    - Se combină aceste soluţii parţiale, obţinând astfel soluţii la problemele de pe niveluri superioare (din arborele de descompunere)

Programare dinamică → TSP

- Dându-se o submulţime S de noduri din V cu $1 \in S$ şi $j \in S$, $j \neq 1$, se consideră $C(S, j)$ lungimea celui mai scurt drum de la nodul 1 la nodul j care trece prin toate nodurile din S

- Observaţii
  - Dacă $|S| = 2$, atunci $C(S, k) = d_{1,k}$ pentru $k = 2, 3, . . . , n$
  - Dacă $|S| > 2$, atunci există $m \in S - \{k\}$ a.î. $C(S, k) =$ lungimea turului optim de la 1 la $m + d_{m,k}$

- Definiţia recusrsivă a soluţiei optime
  - $C(S, k) = d_{1,k}$ dacă $S = \{1, k\}$

    $\min_{m \neq k,\ m \in S} [C(S - \{k\}, m) + d_{m,k}]$, altfel

Programare dinamică → TSP

```
function algorithmTSP(G, n)
    for k = 2 to n do
        C({1, k}, k) := d_{1,k}
    end for
    for s = 3 to n do
        for all S from {1, 2, . . . , n} with |S| = s do
                for all k ϵ S do
                        C(S, k) = min_{m≠k,m ϵ S}[C(S − {k},m) + d_{m,k}]
                        opt := min_{k≠1}[C({1, 2, 3, . . . , n}, k) + d_{1,k}
                end for
         end for
    end for;
    return (opt)
end
```

Programare dinamică → TSP

- Complexitate:
  - Temporală $$(n-1)\sum_{k=1}^{n-3}\binom{n-2}{k}+2(n-1)\sim \mathrm{O}(n^2 2^n) << \mathrm{O}(n!)$$
  - Spaţială $$\sum_{k=1}^{n-1}k\binom{n-1}{k}=(n-1)2^{n-2}\sim \mathrm{O}(n2^n)$$

Problema comisului voiajor – Algoritmi

- Exacţi
  - Forţa brută → alegerea permutării optime
  - Programare dinamică
  - **Branch-and-bound**
  - Programare liniară
- Euristici
  - Constructive
  - De îmbunătăţire

Branch-and-bound

- Principii
  - Căutare ramificată → expandarea "inteligentă" a unui nod din arborele de căutare
  - Căutare mărginită → căutarea se realizează între anumite limite
  - Parcuregerea nodurilor → mod special
    - Nodurile se adaugă într-o coadă de priorităţi
    - Criteriul de ordine pt un nod curent n
      - f(n) = g(n) + h(n), unde
        - g(n) – "distanţa" de la rădăcina arborelui de căutare la nodul curent n → cât a avansat căutarea
        - h(n) – o aproximare a distanţei de la nodul curent n până la soluţia finală → cât mai trebuie căutat
    - Margine inferioară (lower bound)
    - Margine superioară (upper bound)

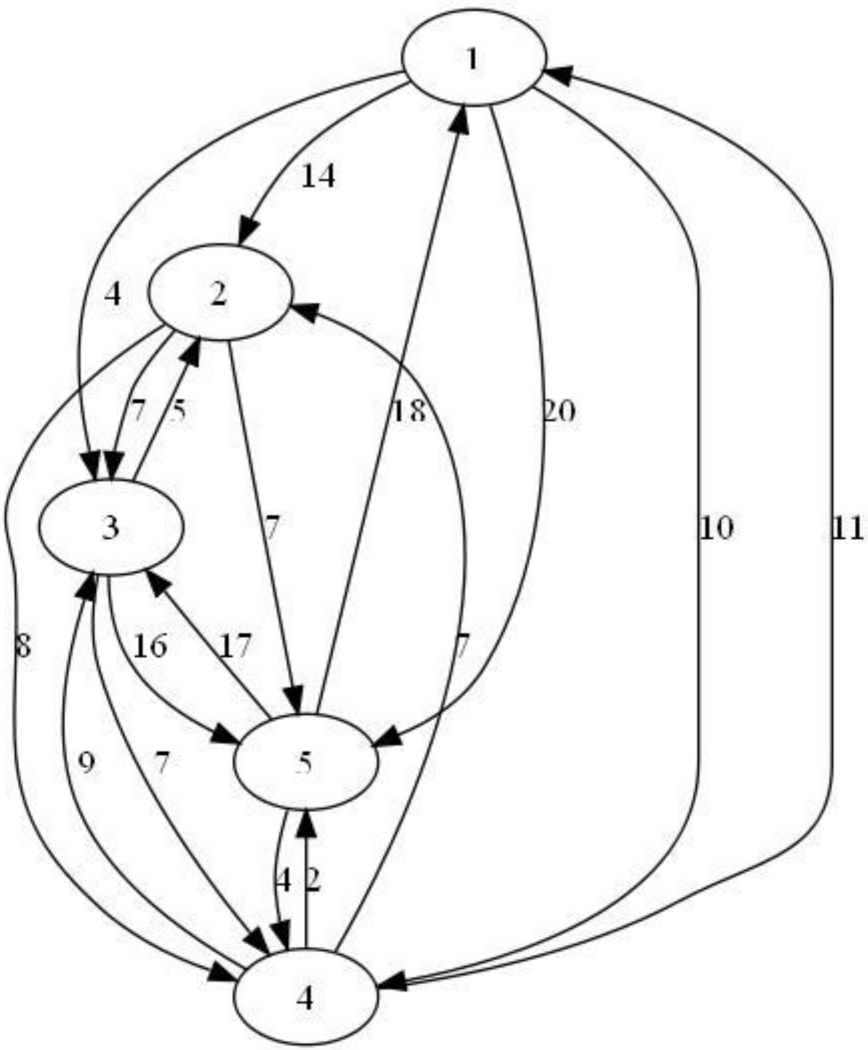
Branch-and-bound → TSP

- Configuraţia iniţială
  - toate muchiile grafului
- Expandarea
  - considerarea (includerea sau nu) unei muchii
- Configuraţia finală
  - ciclul cel mai scurt

Branch-and-bound → TSP

- Lower bound (LB)
  - ½ din lungimea turului format din cei mai apropiaţi 2 vecini ai fiecărui nod
- Condiţii la ramificare
  - Dacă excluderea unei muchii determină apariţia unor noduri cu mai puţin de 2 vecini se renunţă la excludere
  - Dacă adăugarea unei muchii determină apariţia unor noduri cu mai mult de 2 vecini se renunţă la adăugare
  - Dacă LB-ul unui nod-fiu e ≥ LB-ul nodului-părinte, nodul-fiu nu mai merită explorat ("pruned")
  - Dacă avem doi fii de explorat, primul va fi explorat cel cu LB-ul mai mic (coadă de priorităţi)

Branch-and-bound → TSP

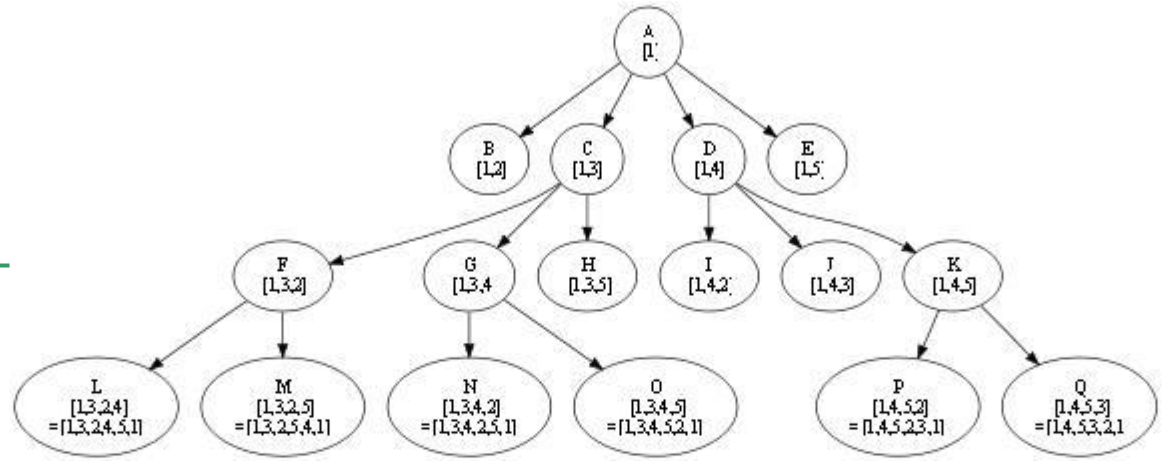


	1	2	3	4	5
1	-	14	4	10	20
2	-	-	7	8	7
3		5	-	7	16
4	11	7	9	-	2
5	18	-	17	4	-

Branch-and-bound → TSP

- Se construieşte turul progresiv
- LB = lungimea turului parţial + muchia cea mai scurtă care iasă din ultimul nod al turului parţial + cele mai scurte muchii care iasă din restul nodurilor (care nu fac parte din turul parţial)
- Turul minim iniţial = ∞
- Dacă LB < turul minim curent → nod promiţător (se depune în coadă)
- Dacă LB ≥ turul minim şi s-a găsit deja un tur potenţial → se renunţă la explorarea căii respective (în arborel de căutare → prune)

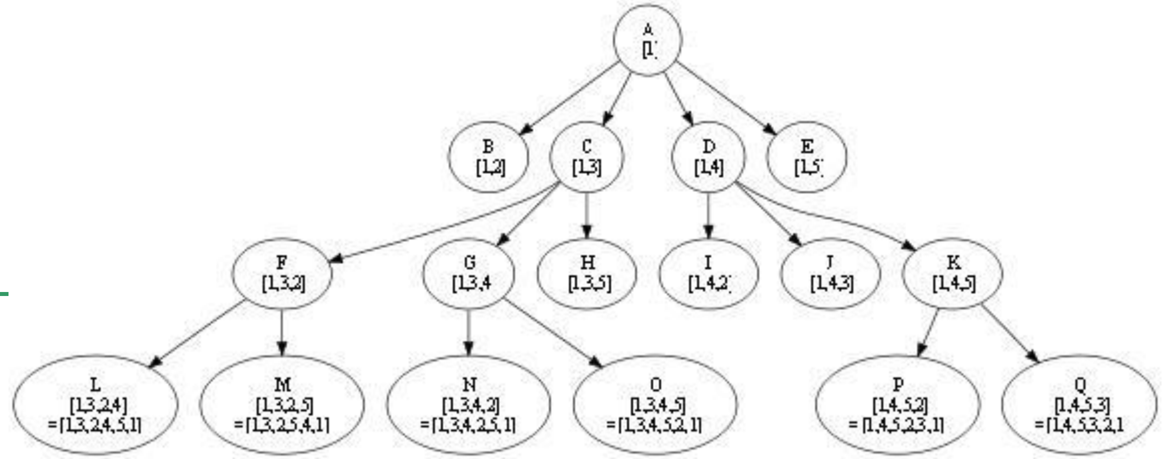
B&B TSP



- A
  - Tur parţial 1
  - LB = 4+(7+5+2+4)=22
  - LB < Tur minim = ∞

	1	2	3	4	5
1	-	14	4	10	20
2	-	-	7	8	7
3		5	-	7	16
4	11	7	9	-	2
5	18	-	17	4	-

B&B TSP



- B
  - Tur parţial 1,2
  - LB = 14+(7+5+2+4)=32
- C
  - Tur parţial 1,3
  - LB = 22
- D
  - Tur parţial 1,4
  - LB = 26
- E
  - Tur parţial 1,5
  - LB = 38

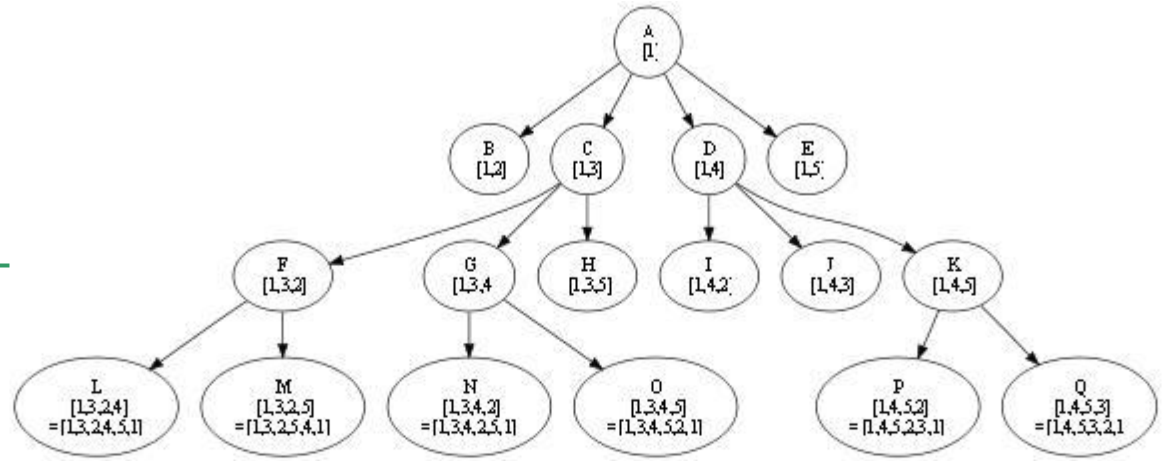
	1	2	3	4	5
1	-	14	4	10	20
2	-	-	7	8	7
3		5	-	7	16
4	11	7	9	-	2
5	18	-	17	4	-

➔ LB minim = 22 ➔ următorul nod explorat: C

➔ Coada: A(22) **C(22)** D(26) B(32) E(38)

B&B TSP

- F
  - Tur parţial 1,3,2
  - LB = 22
- G
  - Tur parţial 1,3,4
  - LB = 24
- H
  - Tur parţial 1,3,5
  - LB = 33

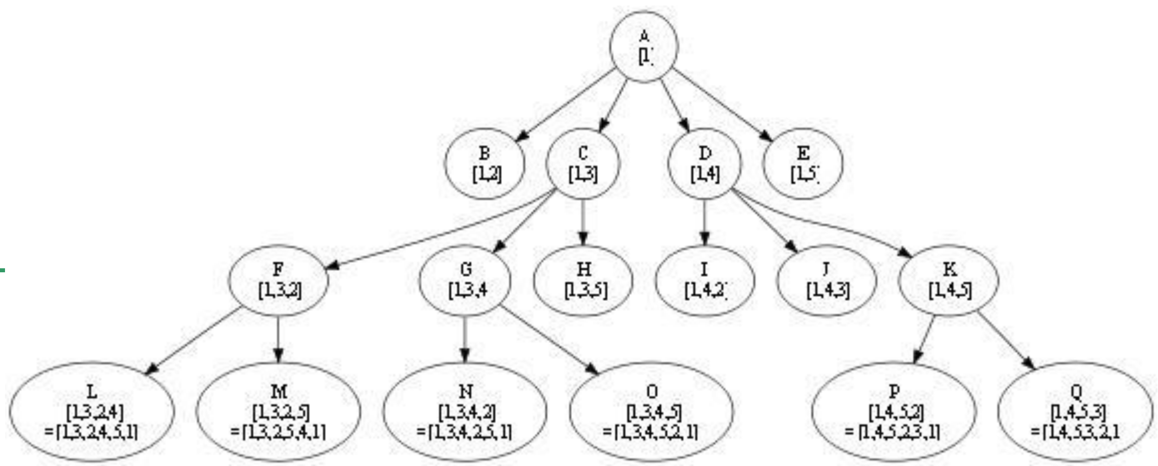


	1	2	3	4	5
1	-	14	4	10	20
2	-	-	7	8	7
3		5	-	7	16
4	11	7	9	-	2
5	18	-	17	4	-

→ LB minim = 22 → următorul nod explorat: F

→ Coada: A(22) C(22) **F(22)** G(24) D(26) B(32) H(33) E(38)

B&B TSP



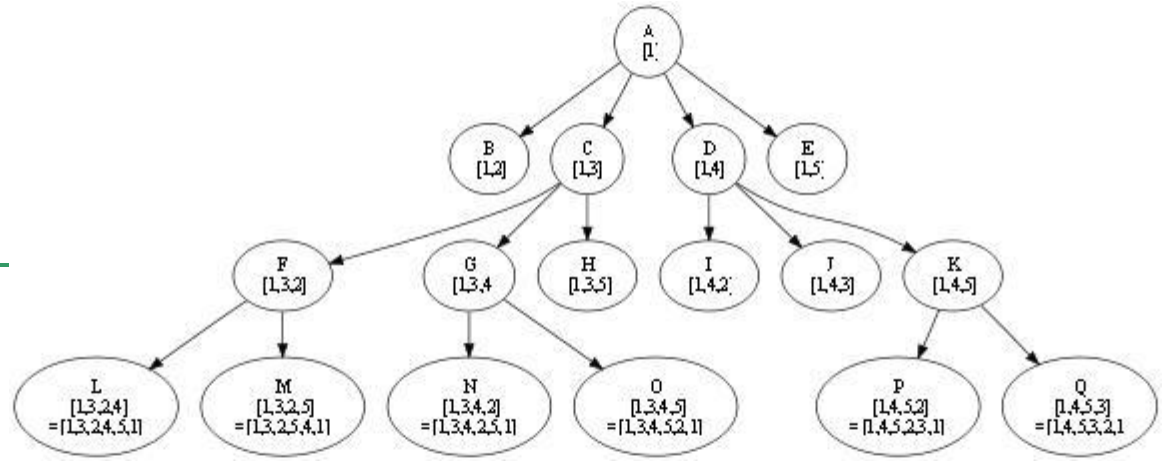
- L
  - Tur parţial 1,3,2,4 = 1,3,2,4,5,1
  - Lungime = 37
- M
  - Tur parţial 1,3,2,5=1,3,2,5,4,1
  - Lungime = 31

	1	2	3	4	5
1	-	14	4	10	20
2	-	-	7	8	7
3		5	-	7	16
4	11	7	9	-	2
5	18	-	17	4	-

➔ LB minim = 23 ➔ următorul nod explorat: G

➔ Coada: A(22) C(22) F(22) **G(23)** D(27) B(32) M(31) H(33) L(37) E(38)

B&B TSP



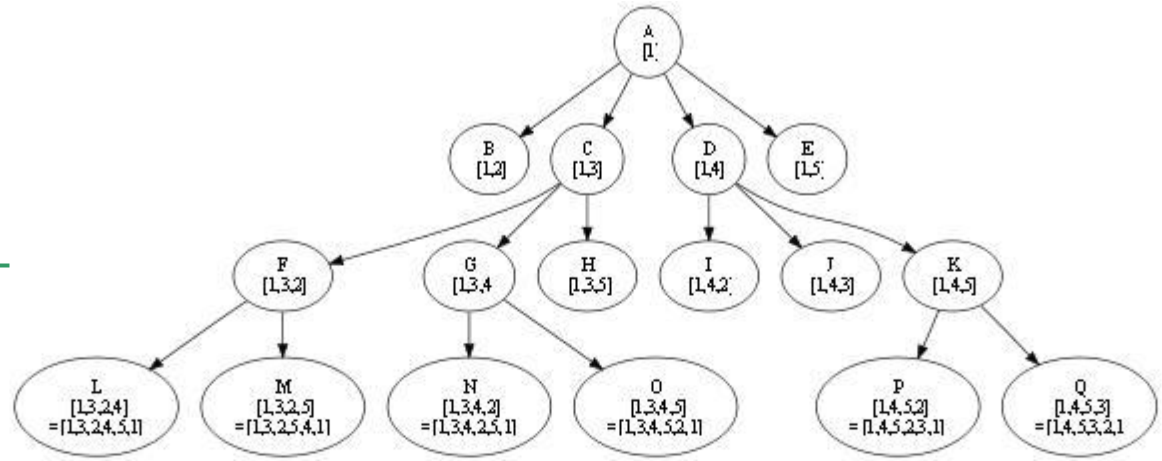
- N
  - Tur parţial 1,3,4,2 = 1,3,4,2,5,1
  - Lungime = 43
- O
  - Tur parţial 1,3,4,5=1,3,4,5,2,1
  - Lungime = -

	1	2	3	4	5
1	-	14	4	10	20
2	-	-	7	8	7
3		5	-	7	16
4	11	7	9	-	2
5	18	-	17	4	-

→ LB minim = 27 → următorul nod explorat: D

→ Coada: A(22) C(22) F(22) G(23) **D(27)** B(32)M(31) H(33) L(37) E(38) N(43)

B&B TSP

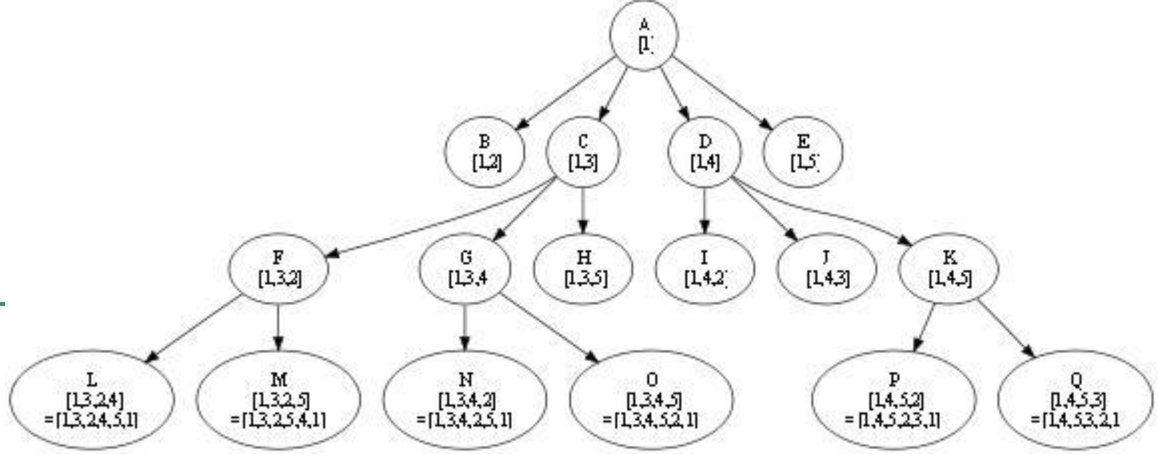


- I
  - Tur parţial 1,4,2
  - LB = 33
- J
  - Tur parţial 1,4,3
  - LB = 35
- K
  - Tur parţial 1,4,5
  - LB = 28

	1	2	3	4	5
1	-	14	4	10	20
2	-	-	7	8	7
3		5	-	7	16
4	11	7	9	-	2
5	18	-	17	4	-

- LB minim = 28 ➔ următorul nod explorat: K
- Coada: A(22) C(22) F(22) G(23) D(27) **K(28)** B(32) I(33) M(31) H(33) J(35) L(37) E(38) N(43)

B&B TSP



- P
  - Tur parţial 1,4,2,5=1,4,2,5,3,1
  - Lungime = -
- Q
  - Tur parţial 1,4,5,3=1,4,5,3,2,1
  - Lungime = -

	1	2	3	4	5
1	-	14	4	10	20
2	-	-	7	8	7
3		5	-	7	16
4	11	7	9	-	2
5	18	-	17	4	-

- ...
- LB minim = 30
- S-a găsit un tur de lungime 30
- B nu mai merită explorat (se elimină din coadă)
- Restul nodurilor nu mai merită cercetate (LB > turul minim)
- Coada: A(22) C(22) F(22) G(23) D(27) K(28) B(32) I(33) M(31) H(33) J(35) L(37) E(38) N(43)

Problema comisului voiajor – Algoritmi

- Exacţi
  - Forţa brută → alegerea permutării optime
  - Programare dinamică
  - Branch-and-bound
  - **Programare liniară**
- Euristici
  - Constructive
  - De îmbunătăţire

Programare liniară - TSP

- http://www.tsp.gatech.edu/methods/dfj/index.html
- http://www.youtube.com/watch?v=-cLsEHP0qt0&feature=channel

Algoritmi

- Exacţi
  - Forţa brută → alegerea permutării optime
  - Programare dinamică
  - Branch-and-bound
  - Programare liniară

- **Euristici**
  - Constructive
  - De îmbunătăţire

Euristici – TSP

- Euristici constructive
  - Cel mai apropiat vecin + greedy → caseStudyJohnson.pdf
  - Local search
  - Algoritmul lui Christofides → spanning tree
- Euristici de îmbunătăţire (improved heuristics)
  - Simulated annealing
  - Tabu search
  - EAs
  - ACO

Euristici – TSP

- Euristici constructive
  - **Cel mai apropiat vecin + greedy** → caseStudyJohnson.pdf
  - Local search
  - Algoritmul lui Christofides → spanning tree

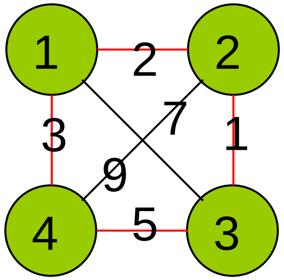
- Euristici de îmbunătăţire (improved heuristics)
  - Simulated annealing
  - Tabu search
  - EAs
  - ACO

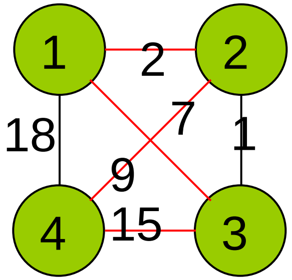
Cel mai apropiat vecin

- Ordonarea crescătoare a muchiilor
- Alegerea celei mai scurte muchii, cu condiţiile
  - orice vârf să aibă maxim 2 vecini
  - să nu se formeze cicluri cu mai puţin de *n* muchii

- Complexitatea
  - O(n^2log n)
  - Folosirea arborilor *k* dimensionali (*k*d trees) şi a unei cozi de priorităţi pentru reţinerea muchiilor ➔ O(n log n)

- Problemă
  - nu găseşte soluţia optimă întotdeauna



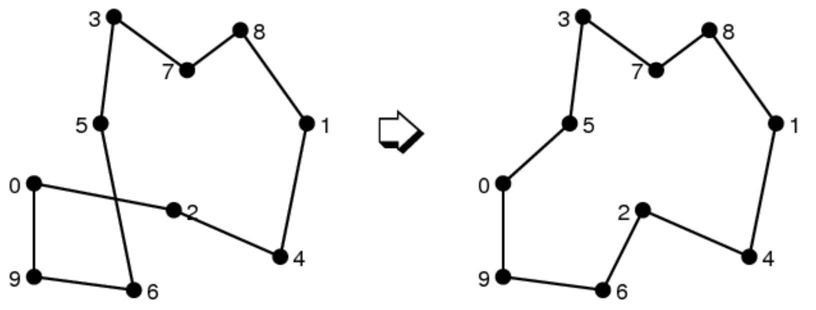


Euristici – TSP

- Euristici constructive
  - Cel mai apropiat vecin + greedy → caseStudyJohnson.pdf
  - **Local search**
  - Algoritmul lui Christofides → spanning tree
- Euristici de îmbunătăţire (improved heuristics)
  - Simulated annealing
  - Tabu search
  - EAs
  - ACO

Căutare locală

- Se porneşte cu un ciclu oarecare
- Se modifică ciclul prin operaţii de
  - schimbare de noduri
    - ABCDEF → AECDBF
  - inserţie de nod
    - ABCDEF → ADBCEF
  - schimbare de k muchii (k = 2)



- în vederea obţinerii unui ciclu mai bun (scurt)

Căutare locală

- Euristici bazate pe
  - schimbul a k elemente
    - noduri
    - muchii
  - Vecinătăţi complexe
    - algoritmul Lin-Kernighan & versiuni

Euristici – TSP

- Euristici constructive
  - Cel mai apropiat vecin + greedy → caseStudyJohnson.pdf
  - Local search
  - **Algoritmul lui Christofides** → spanning tree
- Improved heuristics (euristici de îmbunătăţire)
  - Simulated annealing
  - Tabu search
  - EAs
  - ACO

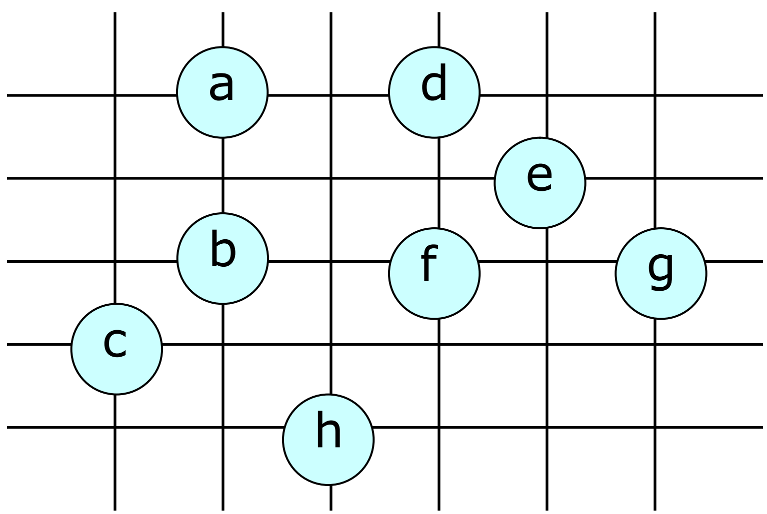
Algoritmul lui Christofides

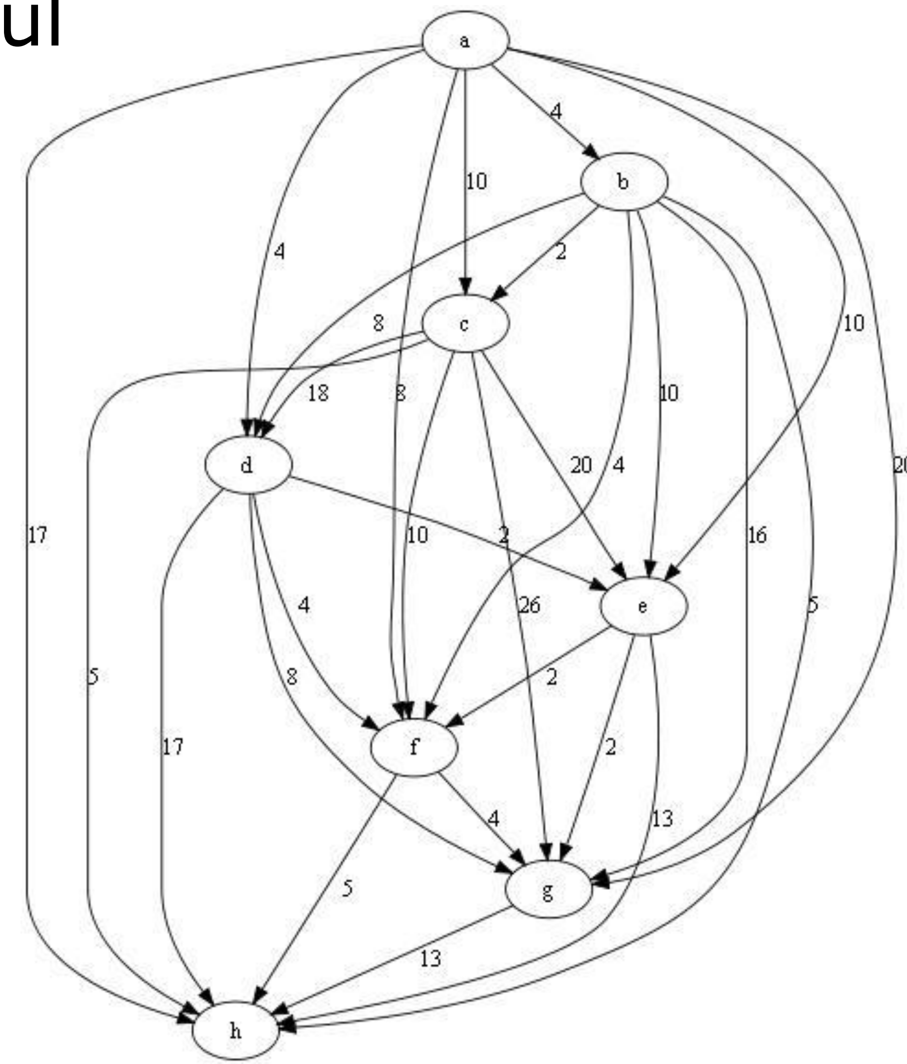
- TSP în graf complet G şi care respectă ingealitatea triunghiului

- Algoritm
  - Se creează arborele de acoperire minimă (A) al lui G
  - Notând cu I mulţimea nodurilor din A de grad impar, se găseşte o potrivire perfectă P cu muchii de lungimi minime în graful G peste nodurile din I
  - Se combină muchiile din P şi A formând un multigraf H
  - Se formează un circuit Eulerian în H (H este Eulerian pt că este conect şi format doar din noduri de grad par)
  - Se transformă circuitul Eulerian într-unul Hamiltonian prin "sărirea" nodurilor deja vizitate (*shortcutting*).

Algoritmul lui Christofides

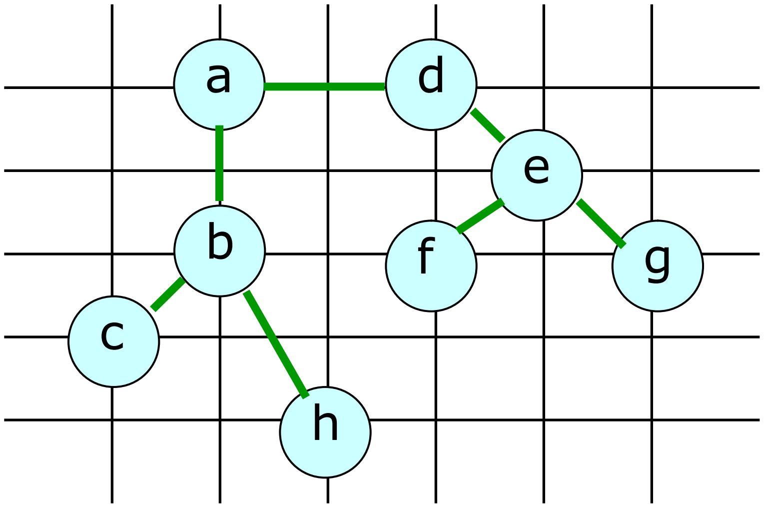
- Presupunem următorul graf cu distanţe Euclidiene între noduri





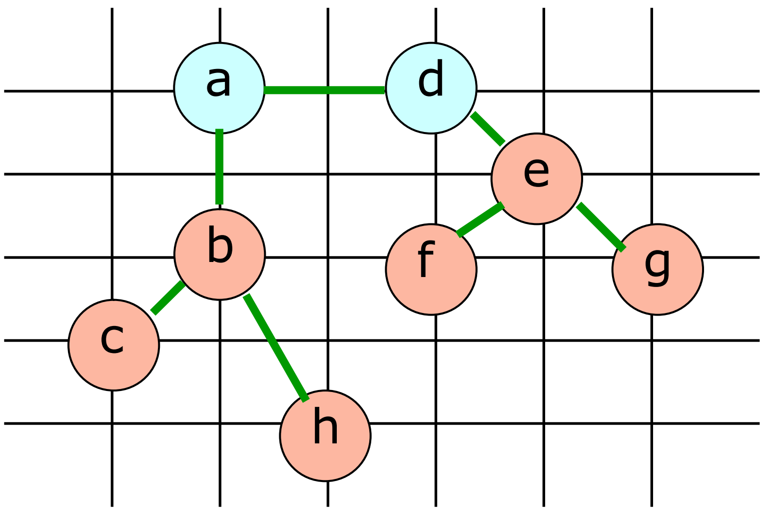
Algoritmul lui Christofides

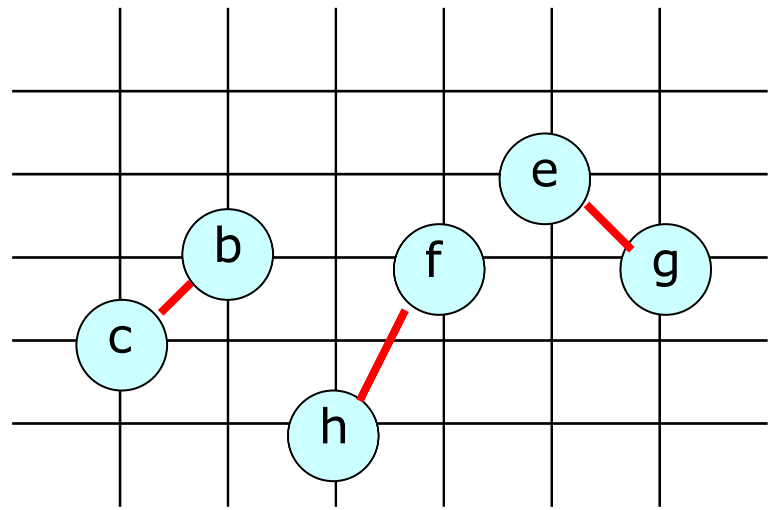
- Se creează arborele de acoperire minimă (**A**) al lui G
  - Algoritmul lui Prim
    - Se pleacă dintr-un nod şi se aleg pe rand cei mai apropiaţi vecini ai nodurilor deja vizitate a.î. să nu se formeze cicluri până se ajunge la o componentă conexă
  - Algoritmul lui Kruskal
    - Se aleg pe rând muchiile de cost minim a.î. să nu se formeze cicluri până se ajunge la o componentă conexă



Algoritmul lui Christofides

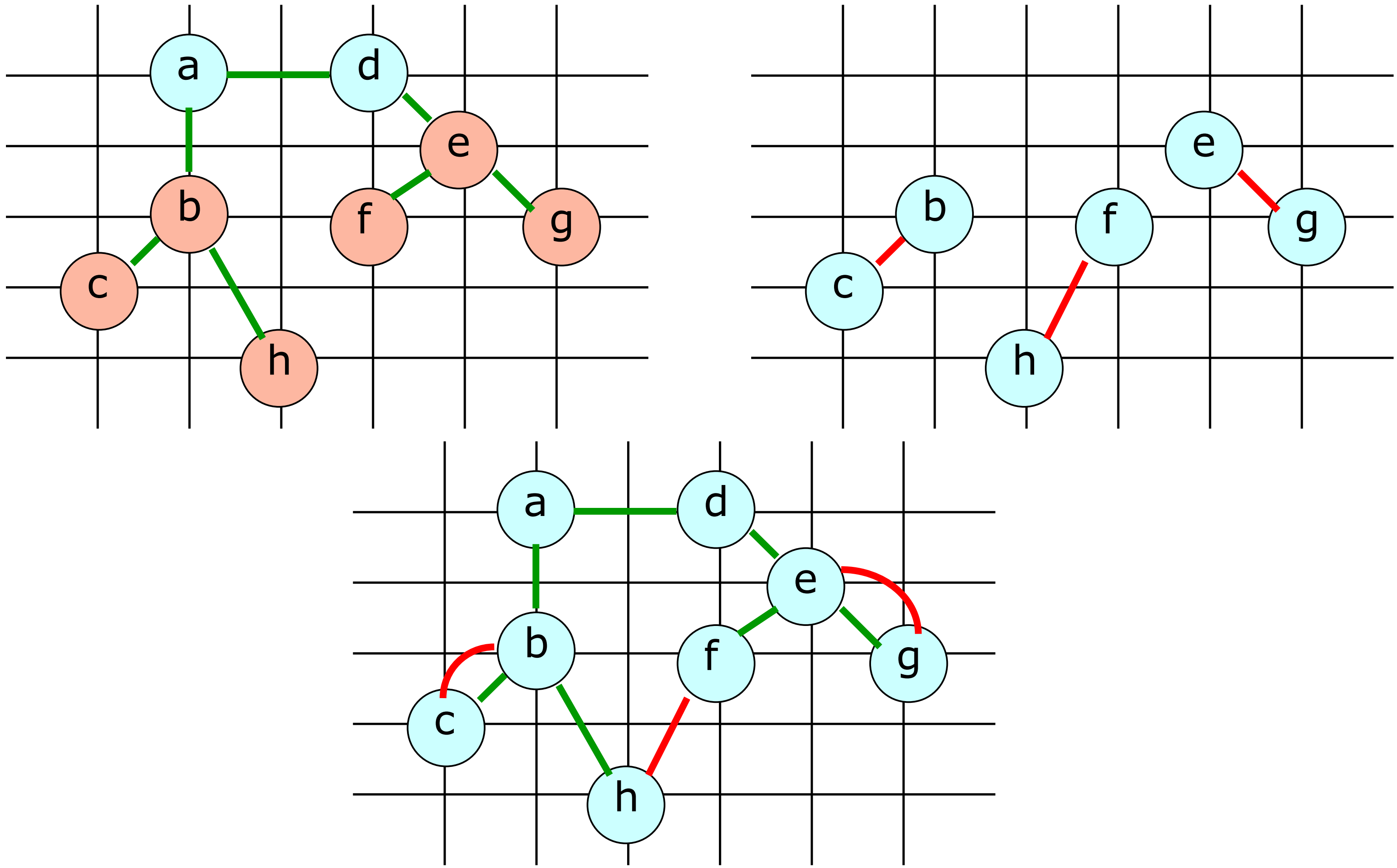
- Notând cu I mulţimea nodurilor din **A** de grad impar, se găseşte o potrivire perfectă **P** cu muchii de lungimi minime în graful G peste nodurile din I





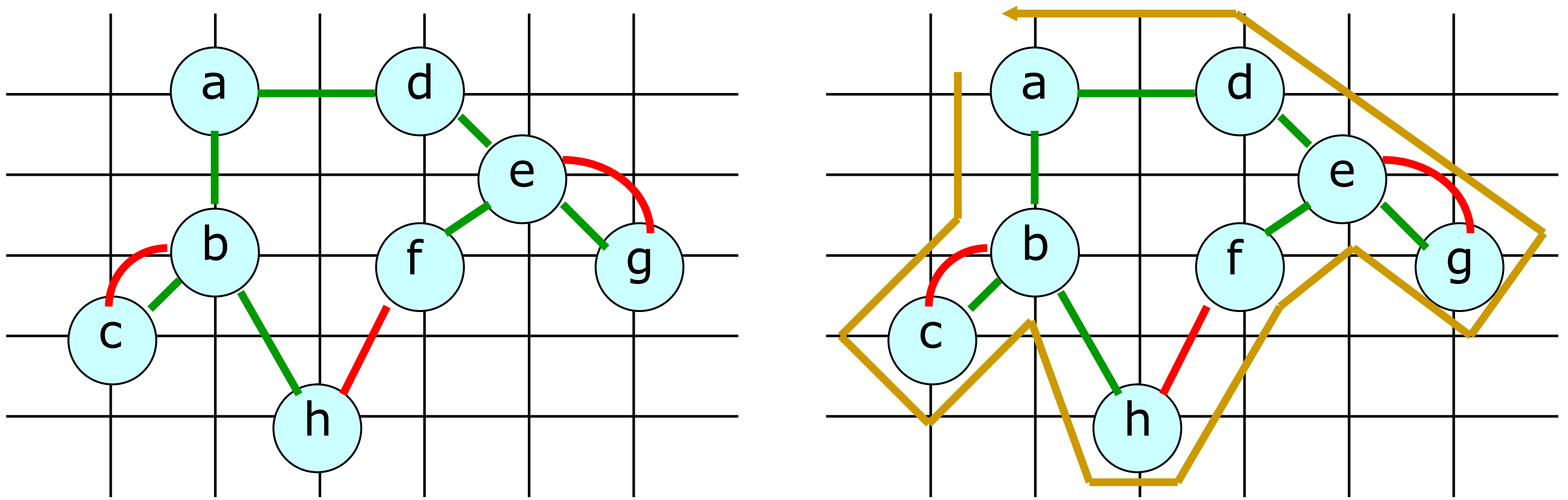
Algoritmul lui Christofides

- Se combină muchiile din **P** şi **A** formând un multigraf H



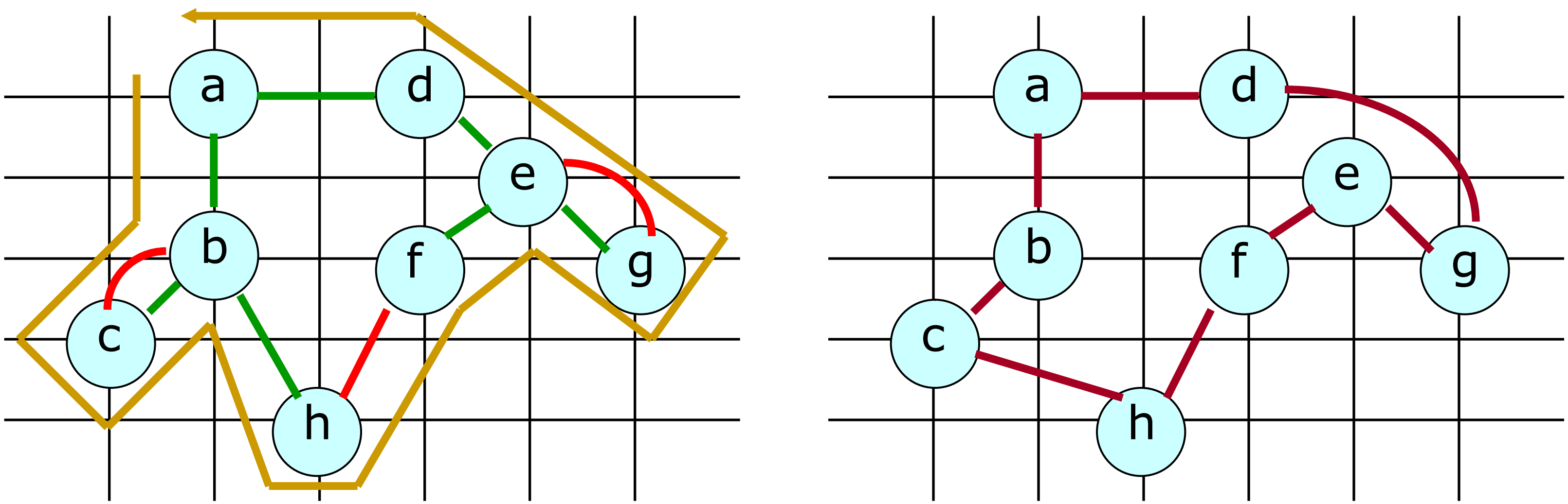
Algoritmul lui Christofides

- Se formează un circuit Eulerian în H (H este Eulerian pt că este conect şi format doar din noduri de grad par)



Algoritmul lui Christofides

- Se transformă circuitul Eulerian într-unul Hamiltonian prin “săririea” nodurilor deja vizitate (*shortcutting*).



Euristici – TSP

- Euristici constructive
  - Cel mai apropiat vecin + greedy → caseStudyJohnson.pdf
  - Local search
  - Algoritmul lui Christofides → spanning tree
- Improved heuristics (euristici de îmbunătăţire)
  - Simulated annealing
  - Tabu search
  - EAs
  - ACO

Euristici – TSP

- Euristici constructive
  - Cel mai apropiat vecin + greedy → caseStudyJohnson.pdf
  - Local search
  - Algoritmul lui Christofides → spanning tree
- Improved heuristics (euristici de îmbunătăţire)
  - **Simulated annealing**
  - Tabu search
  - EAs
  - ACO

Simulated annealing

- Ideea de bază:
  - Acceptarea noii ponteţiale soluţii se face cu o anumită probabilitate
- Sursa de inspiraţie:
  - Procesul de reorganizare a structurii unui solid supus unui tratament termic
    - Când solidul se încălzeşte (prin topire) particulele se distribuie aleator
    - Când solidul se răceşte particulele se reorganizează ajungând în configuraţii cu energii tot mai mici
- Alte denumiri:
  - Tratament termic simulat, călire simulată
- Metodă propusă de
  - Kirkpatrick, Gelatt, Vecchi (1983), Cerny (1985)

Simulated annealing - algoritm

- Iniţializare x(0)
- K := 0

```
Repetă
    generare vecin x’ al lui x(k)
    Dacă f(x’) < f(x(k))
      atunci x(k+1) := x’
      altfel  x(k+1) := x’ cu probabilitatea exp(-(f(x’)-f(x(k))/T)
    recalculează T
    k := k + 1
Până când este satisfăcută o condiţie de oprire
soluţie x* := x(k -1)
```

Unde T (temperatura) este un parametru de control al procesului de optimizare

Simulated annealing - algoritm

- Iniţializare x(0)
- K := 0

```
Repetă
   generare vecin x’ al lui x(k)
   Dacă f(x’) < f(x(k))
     atunci x(k+1) := x’
     altfel
            u := random(0, 1)
            dacă u < P(x(k+1)=x’)=1/(1+exp(-(f(x’)-f(x(k))/T))
                   atunci x(k+1) := x’
                   altfel x(k+1) := x(k)
    recalculează T
   k := k + 1
Până când este satisfăcută o condiţie de oprire
Soluţie x* := x(k -1)
```

Simulated annealing

T – mare ➔ probabilitate mare de acceptare a unei configuraţii noi (căutare aleatoare)

T – mică ➔ probabilitate mare de acceptare doar a configuraţiilor care îmbunătăţesc funcţia obiectiv

Scheme de răcire:

$T(k) = T(0) / (k + 1)$

$T(k) = T(0) / \ln(k + c)$

$T(k) = a\, T(k-1)$, cu $a < 1$

$T(0)$ – de obicei se alege mare

Simulated annealing – TSP

- Soluţia iniţială
  - un ciclu Hamiltonian (o permutare a celor n oraşe)
- Vecinătate
  - Transformare 2-opt a unei permutări
  - x(k) = ABCFEDG → ABCFEDG → ABCDEFG = x′
- Funcţia obiectiv
  - Costul asociat unui ciclu
- Schema de răcire a temperaturii
  - $T(k) = T(0) / (1 + \log(k))$

Euristici – TSP

- Euristici constructive
  - Cel mai apropiat vecin + greedy → caseStudyJohnson.pdf
  - Local search
  - Algoritmul lui Christofides → spanning tree
- Improved heuristics (euristici de îmbunătăţire)
  - Simulated annealing
  - **Tabu search**
  - EAs
  - ACO

Tabu search

- O căutare locală ghidată printr-o memorie flexibilă
  - Soluţia următoare este cea mai bună vecină a soluţiei curente
  - Chiar dacă s-a găsit un optim local se permite căutarea unei noi soluţii
    - ➔ ciclarea soluţiilor – rezolvată prin folosirea unei liste tabu
      - Previne re-explorarea unei soluţii anterioare
      - Previne execuţia unei mutări de 2 ori
- Nu există elemente stochastice (ca la Simulated Annealing)

Tabu search

- Iniţializare x(0)
- $x^*=x^{best}=x(0)$
- K = 0
- T =Ø

```
Repetă
    dacă există vecini ai lui x(k) care nu sunt tabu
      atunci x' = cel mai bun vecin al lui x(k) care nu e tabu
      x(k+1) := x'
      Dacă f(x') < f(x*)
                  atunci x* := x'
      k := k + 1
      update tabu list
    altfel stop
Până când este satisfăcută o condiţie de oprire
Soluţie x*
```

Tabu search – TSP

- Soluţia iniţială
  - un ciclu Hamiltonian (o permutare a celor n oraşe)
- Vecinătate
  - Transformare 2-opt a unei permutări
  - x(k) = ABCFEDG → ABCFEDG → ABCDEFG = x’
- Funcţia obiectiv
  - Costul asociat unui ciclu
- Lista tabu
  - Muchiile noi (2) care au intrat în soluţie într-o iteraţie
  - Muchiile care au intrat in lista tabu nu pot fi eliminate din soluţie

Euristici – TSP

- Euristici constructive
  - Cel mai apropiat vecin + greedy → caseStudyJohnson.pdf
  - Local search
  - Algoritmul lui Christofides → spanning tree
- Improved heuristics (euristici de îmbunătăţire)
  - Simulated annealing
  - Tabu search
  - **EAs**
  - ACO

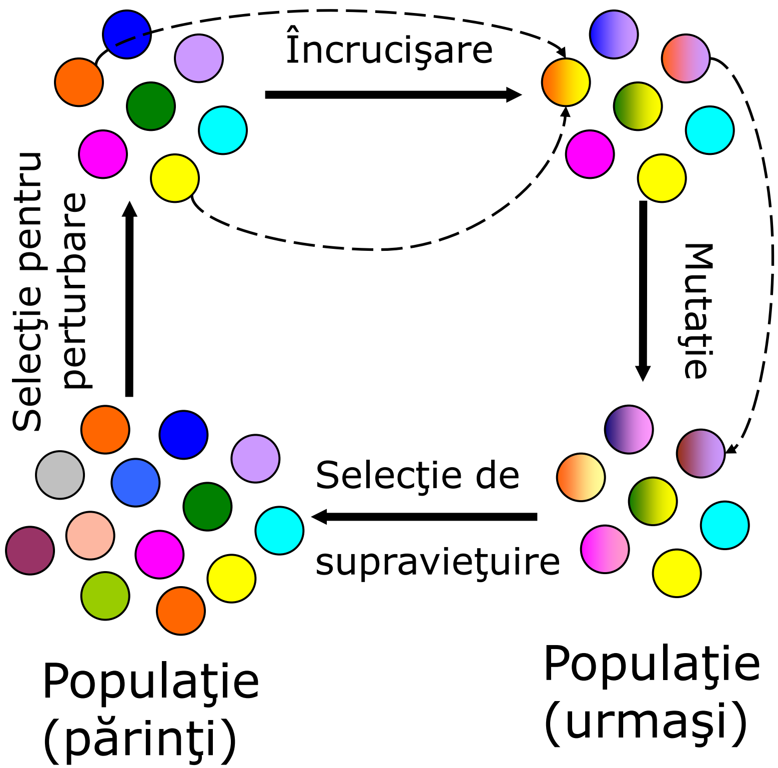
Algoritmi evolutivi

- Algoritmi
  - Inspiraţi din natură (biologie)
  - Iterativi
  - Bazaţi pe
    - populaţii de potenţiale soluţii
    - căutare aleatoare ghidată de
      - Operaţii de selecţie naturală
      - Operaţii de încrucişare şi mutaţie
  - Care procesează în paralel mai multe soluţii
- Metafora evolutivă

Evoluţie naturală	Rezolvarea problemelor
Individ	Soluţie potenţială (candidat)
Populaţie	Mulţime de soluţii
Cromozom	Codarea (reprezentarea) unei soluţii
Genă	Parte a reprezentării
Fitness (măsură de adaptare)	Calitate
Încrucişare şi mutaţie	Operatori de căutare
Mediu	Spaţiul de căutare al problemei

Algoritmi evolutivi

```
Initializare populaţie P(0)
Evaluare P(0)
g := 0; //generaţia
CâtTimp (not condiţie_stop) execută
    Repetă
      Selectează 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 o* în P(g+1)
    Până când P(g+1) este completă
    g := g + 1
Sf CâtTimp
```



Algoritmi evolutivi → TSP

- Reprezentare
  - Cromozomul = o permutare de n elemente
- Fitness
  - Lumgimea ciclului codat de permutare
- Iniţializare
  - Permutarea identică + interschimbări de elemente
- Încrucişare
  - Cu punct de tăietură + corecţii
  - Operatorul PMX → Goldberg (*Alleles*, *loci*, and the TSP)
- Mutaţie
  - Interschimbare de elemente

Algoritmi evolutivi → TSP

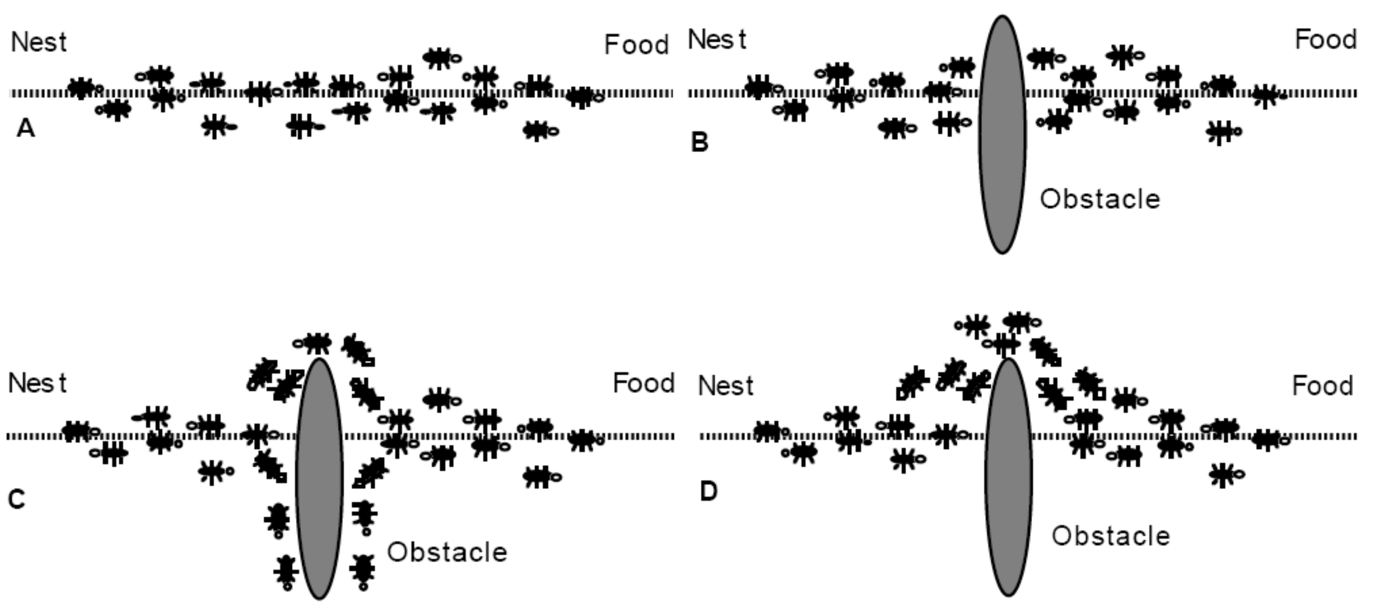
- Reprezentare
  - Cromozomul = un arbore care codează o euristică pentru alegerea următorului nod vizitat
- Fitness
  - Lumgimea ciclului contruit prin folosirea euristicii codată în cromozom
- Iniţializare
  - Arbore corect
- Încrucişare
  - Cu punct de tăietură
- Mutaţie
  - Modificarea unui terminal sau a unei sub-expresii

Euristici – TSP

- Euristici constructive
  - Cel mai apropiat vecin + greedy → caseStudyJohnson.pdf
  - Local search
  - Algoritmul lui Christofides → spanning tree
- Improved heuristics (euristici de îmbunătăţire)
  - Simulated annealing
  - Tabu search
  - EAs
  - **ACO**

ACO

- Preferinţa pentru drumuri cu nivel ridicat de feromon
- Pe drumurile scurte feromonul se înmulţeşte
- Furnicile comunică pe baza urmelor de feromon



ACO → TSP

- Algoritm
  - m furnicuţe sunt plasate în r oraşe alese aleator
  - la fiecare pas, furnicile se deplasează într-un oraş nou
    - modificând feromonul de pe drumul (muchia) respectiv(ă) – modificare locală a urmei
    - memorând oraşele prin care au trecut
  - alegerea noului oraş se bazează pe
    - cantitatea de feromon de pe drumul care urmează a fi parcurs → DP
    - importanţa feromonui de pe drumul care urmează a fi parcurs → DP
    - lungimea drumului care urmează a fi parcurs → IP
  - când toate furnicuţele au trecut prin toate oraşele, furnicuţa care a parcurs cel mai scurt drum mai modifică o dată feromonum de pe drumul ei – modificarea globală a urmei → recompensarea ciclurilor scurte

Cursul următor

- Instruire automata (Machine Learning - ML)
  - introducere in domeniul ML
  - tipuri de probleme
  - metodologia rezolvării unei probleme cu ajutorul unui algoritm de ML
  - aprecierea performanţelor unui algoritm de ML

- De citit:
  - S.J. Russell, P. Norvig – Artificial Intelligence - A Modern Approach → capitolul 18, 19, 20
  - Documenfptele din directoarele: ML, classification, clustering