Curs 10.

Clustering (gruparea datelor biologice)

Biblio: cap. 10 din “An introduction to Bioinformatics algorithms”, N.Jones, P. Pevzner

Slides E. Rouchka

Clustering

- Motivație
- Principiul grupării datelor
- Măsuri de similaritate/disimilaritate
- Metode ierarhice
- Metode partiționale
- Biclustering
- Alte metode

Motivatie

- Analiza comparativă a structurii secvențelor nu permite întotdeauna să se identifice funcția pe care o are o anumită genă

- O variantă ar fi ca în loc să se facă o analiză structurală să se realizeze o analiză funcțională; două gene se consideră a fi similare nu neapărat dacă secvențele lor sunt similare ci dacă se "manifestă" similar

- O genă se "manifestă" ("exprimă") în momentul în care devine activă iar prin măsurarea nivelului de exprimare (gradului de activitate) se poate determina așa-numita expresie genică (gene expression)

Motivatie

- Gradul de activitate al unei gene poate fi măsurat prin cantitatea de ARN mesager corespunzător
- Aceasta se măsoară prin experimente ce folosesc dispozitive de tip “microarray”
- Măsuratorile se efectuează la momente succesive de timp (în condiții experimentale diferite) astfel că fiecărei gene i se poate asocia un “pattern” al activității sale (expresia genică) reprezentat de un vector de valori numerice

Microarray chips [Wikipedia]

Motivatie

- Două gene cu “patternuri” similare de activitate pot fi considerate a avea funcții similare
- Datele înregistrate experimental sunt de regulă specificate prin matrici de intensitate

- Fiecare genă are asociat un vector de valori numerice asociate nivelului de exprimare a genei măsurat la diferite momente de timp

- Gruparea vectorilor conținând nivelele de expresie genică în funcție de distanțele dintre ei permit identificarea unor grupuri (clustere) de gene cu funcționalitate similară

	Timp X	Timp Y	Timp Z
Gena 1	10	8	10
Gena 2	10	0	9
Gena 3	4	8.6	3
Gena 4	7	8	3
Gena 5	1	2	3

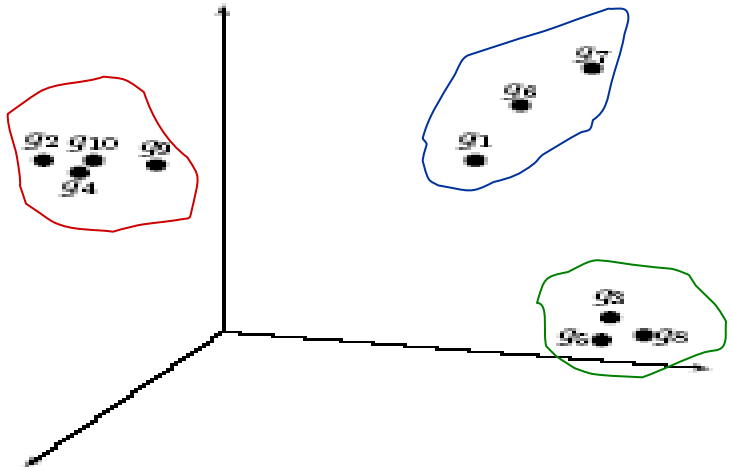
Motivatie

Time	1 hr	2 hr	3 hr
g_1	10.0	8.0	10.0
g_2	10.0	0.0	9.0
g_3	4.0	8.5	3.0
g_4	9.5	0.5	8.5
g_5	4.5	8.5	2.5
g_6	10.5	9.0	12.0
g_7	5.0	8.5	11.0
g_8	2.7	8.7	2.0
g_9	9.7	2.0	9.0
g_{10}	10.2	1.0	9.2

(a) Intensity matrix, I

	g_1	g_2	g_3	g_4	g_5	g_6	g_7	g_8	g_9	g_{10}
g_1	0.0	8.1	9.2	7.7	9.3	2.3	5.1	10.2	6.1	7.0
g_2	8.1	0.0	12.0	0.9	12.0	9.5	10.1	12.8	2.0	1.0
g_3	9.2	12.0	0.0	11.2	0.7	11.1	8.1	1.1	10.5	11.5
g_4	7.7	0.9	11.2	0.0	11.2	9.2	9.5	12.0	1.6	1.1
g_5	9.3	12.0	0.7	11.2	0.0	11.2	8.5	1.0	10.6	11.6
g_6	2.3	9.5	11.1	9.2	11.2	0.0	5.6	12.1	7.7	8.5
g_7	5.1	10.1	8.1	9.5	8.5	5.6	0.0	9.1	8.3	9.3
g_8	10.2	12.8	1.1	12.0	1.0	12.1	9.1	0.0	11.4	12.4
g_9	6.1	2.0	10.5	1.6	10.6	7.7	8.3	11.4	0.0	1.1
g_{10}	7.0	1.0	11.5	1.1	11.6	8.5	9.3	12.4	1.1	0.0

(b) Distance matrix, d



(c) Expression patterns as points in three-dimentsional space.

Principiul gruparii datelor

Scop:
Identificarea de grupuri (clustere) de date caracterizate prin:

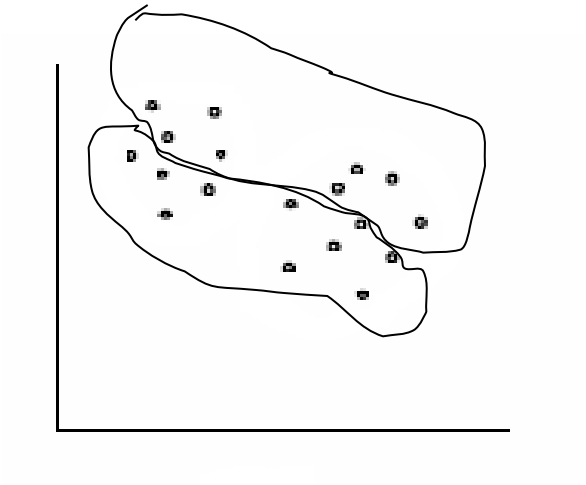
- datele din același grup sunt suficient de similare între ele (criteriul de omogeneitate sau compacitate); gradul de compacitate se măsoară prin suma pătratelor distanțelor dintre elementele unui cluster (within sum of squares)

- datele din grupuri diferite sunt suficient de diferite între ele (criteriul de separabilitate); se măsoară prin suma pătratelor distanțelor dintre date aparținând la două clustere diferite (between sum of squares)

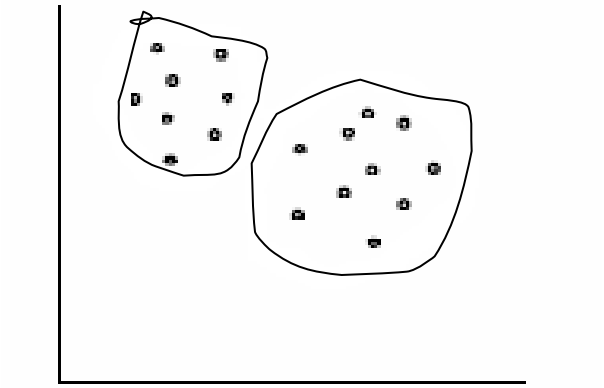
Specific:
Identificarea claselor este o problemă dificilă întrucât nu există o soluție unică și cele două criterii de mai sus sunt conflictuale

Principiul gruparii datelor

Exemplu



Varianta de grupare ce încalcă proprietățile de omogeneitate și separabilitate



Variantă mai naturală de grupare

Masuri de similaritate si disimilaritate

Ipoteze:

- datele sunt specificate prin vectori cu N componente numerice
- uneori vectorii sunt normalizați

Măsură de similaritate:

măsura cosinus: $s(X,Y)=X^TY/(||X||||Y||)$

Măsură de disimilaritate:

distanța euclidiană $d(X,Y)=||X-Y||$

Masuri de similaritate si disimilaritate

Distanțe între două clustere C1 și C2

Single link: cea mai mică distanță dintre două elemente aparținând celor două clustere (un element din C1 și celălalt din C2)

Complete link: cea mai mare distanță dintre două elemente aparținând celor două clustere (un element din C1 și celălalt din C2)

Average link: distanța medie dintre toate perechile de elemente în care un element este în C1 și celălalt în C2

Centroid link: distanța dintre centroizii corespunzători celor două clustere

Alte notiuni

Centroid = valoarea medie a elementelor unui cluster (centrul de greutate al clusterului; nu face neapărat parte din setul de date)

Medoid = elementul din setul de date cel mai apropiat de centroid

Raza cluster = abaterea standard a datelor din cluster (distanța medie față de centroid)

Diametru cluster = media (sau maximul) distanțelor dintre toate perechile de date din cluster

Metode de grupare

Tipuri de metode:

a) In funcție de rezultatul furnizat

ierarhice (aglomerative, divizive) -> produc o ierarhie de partiții

partiționale -> produc o singură partiție

b) In funcție de principiul pe care se bazează:

metode bazate pe similaritate/disimilaritate

metode bazate pe grafuri

metode bazate pe modele probabiliste

metode bazate pe analiza densității datelor

c) In funcție de proprietățile datelor

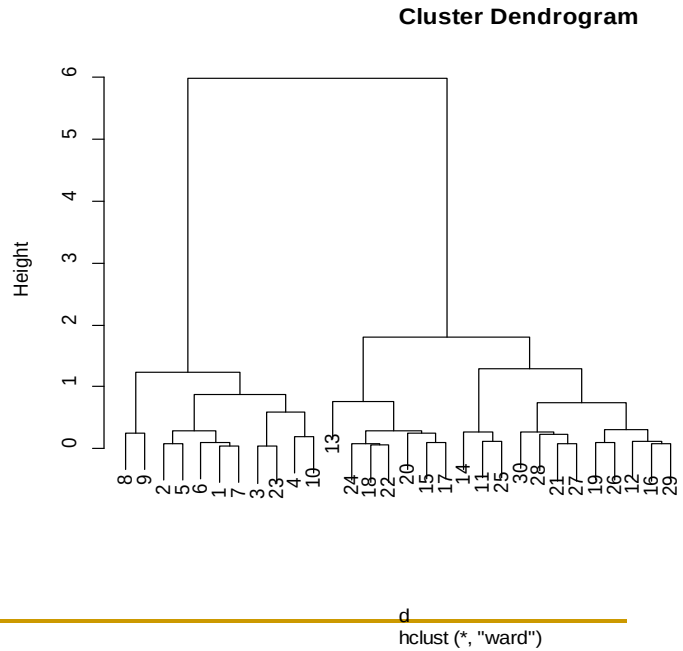
specifice datelor nominale

specifice volumelor mari de date

Metode ierarhice

Specific:

- se construiește o ierarhie de clustere ce poate fi reprezentată printr-o structură arborescentă numită dendrogramă
- rădăcina structurii corespunde cazului în care toate datele sunt într-un singur cluster
- frontiera arborelui corespunde cazului în care fiecare cluster conține un singur element
- fiecare nivel al arborelui corespunde unei partiționări a datelor în clustere



Metode ierarhice

Construirea arborelui se poate face:

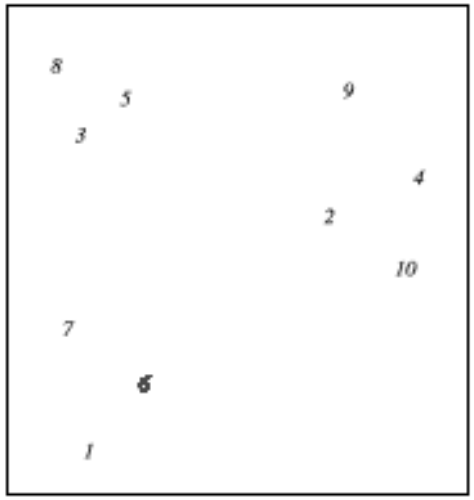
- de la rădăcină către frunze: metoda divizivă
  - la fiecare etapă un cluster este divizat în alte două clustere
  - selecția clusterului de divizat se bazează pe varianța datelor din cadrul lui (se alege clusterul pentru care varianța este maximă)

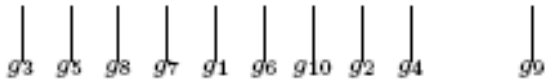
- de la frunze către rădăcină: metoda aglomerativă
  - la fiecare etapă două clustere sunt selectate pentru a fi reunite
  - selecția clusterelor se bazează pe distanța dintre ele; la fiecare etapă se selectează:
    - cele mai apropiate două clustere (folosind una dintre măsurile: single, complete, average)
    - sau clusterele a căror reunire conduce la cea mai mică creștere a variației în cadrul clusterelor (corespunde metodei Ward)

Metode ierarhice

Exemplu:

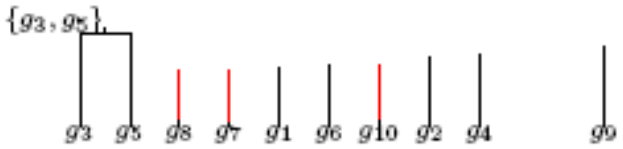
- Varianta aglomerativă

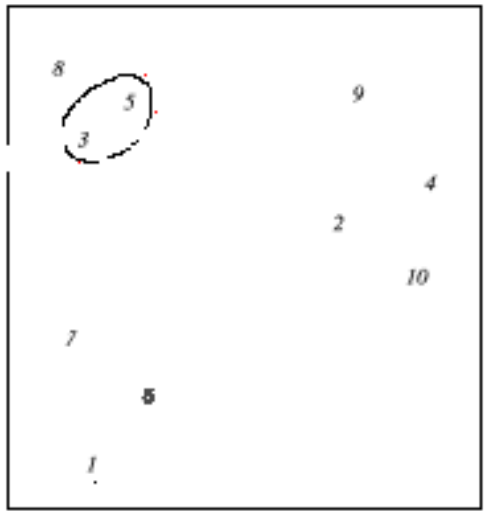




Metode ierarhice

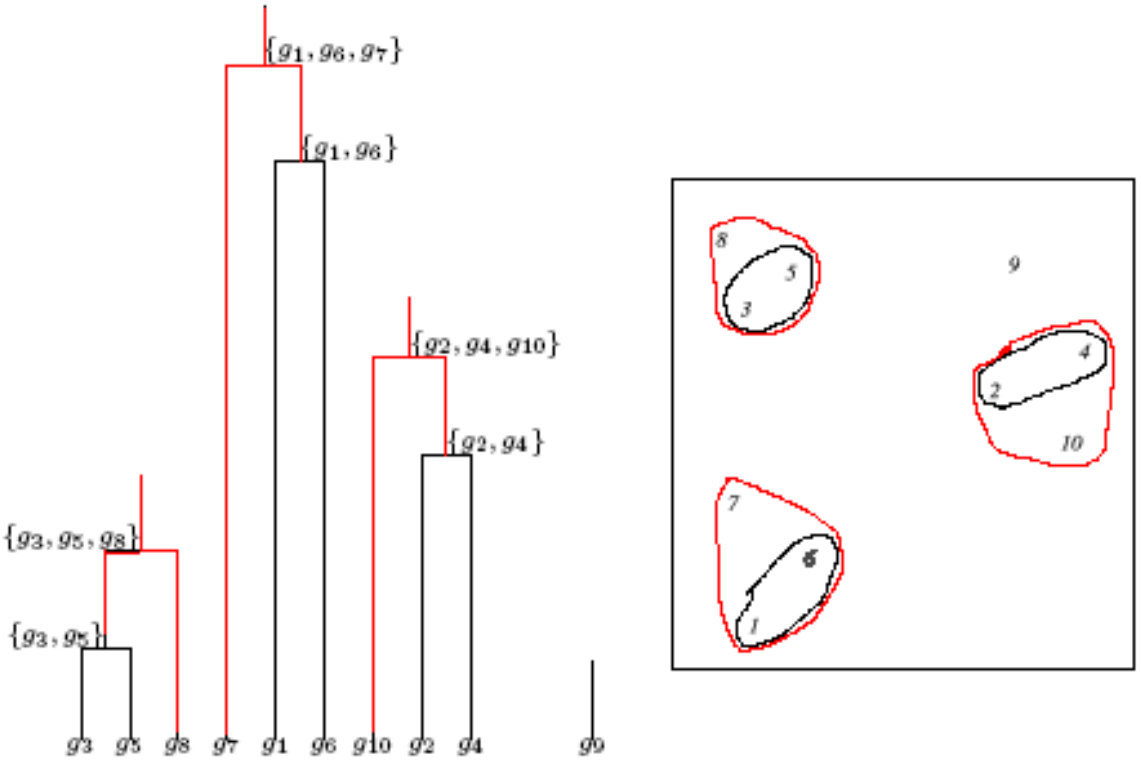
Exemplu:
- Varianta aglomerativă





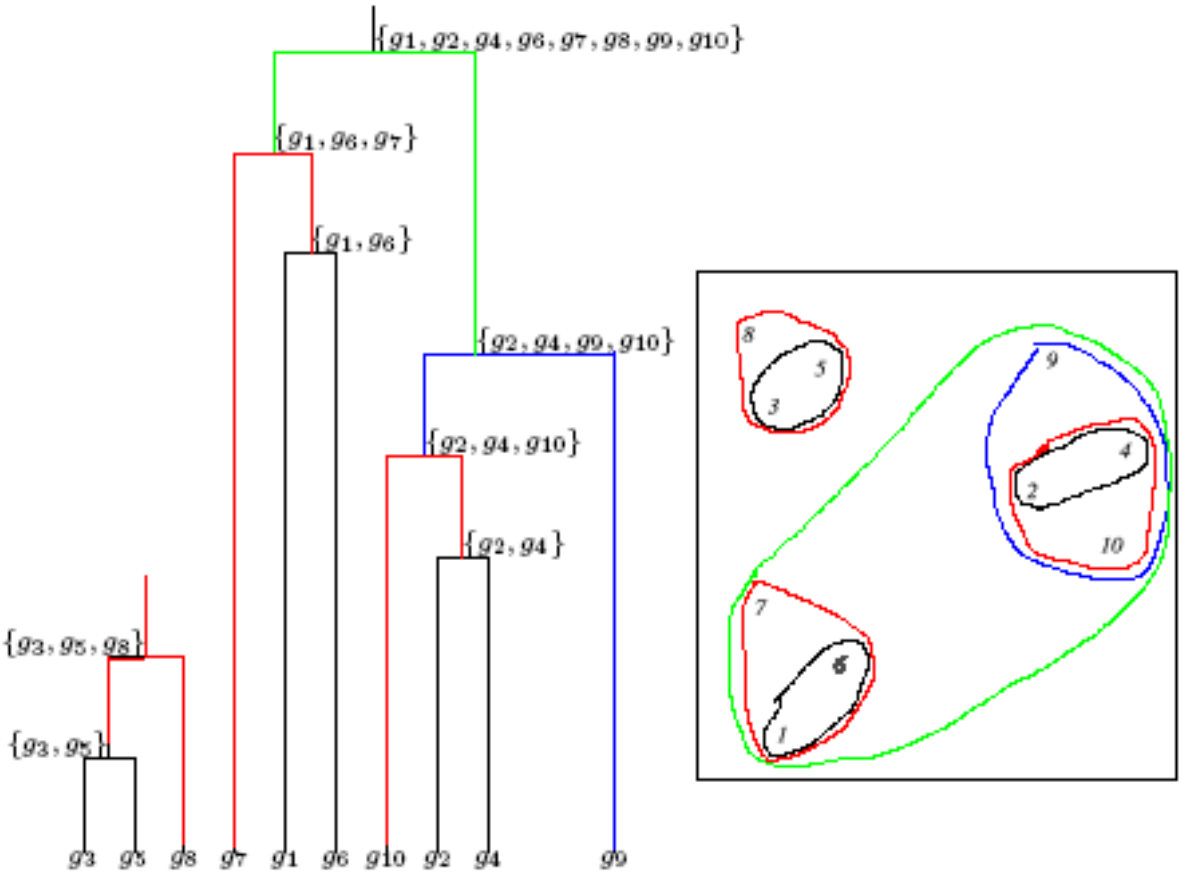
Metode ierarhice

Exemplu:
- Varianta aglomerativă



Metode ierarhice

Exemplu:
- Varianta aglomerativă



Metode ierarhice

Construire dendrograma (varianta aglomerativă):

- se pornește de la matricea de disimilaritate dintre date
  (nxn – în cazul a n date)
- se identifică cele mai apropiate date și se grupează într-un prim cluster
- se calculează distanța dintre noul cluster și celelalte date/clustere și se actualizează matricea de disimilaritate – se elimină liniile corespunzătoare datelor/clusterelor selectate pentru reunire și se introduce o linie corespunzătoare noului cluster (devine matrice (n-1)x(n-1))
- se repetă procedeul identificând la fiecare etapă cele mai apropiate două clustere, reunindu-le și actualizând matricea de disimilaritate

Metode ierarhice

Exemplu (varianta bazată pe reunirea celor mai similare clustere)

Aglomerativ (*n*)

Construiește n clustere având fiecare câte un element
Construiește matricea D de disimilaritate dintre clustere
Inițializează arborele A cu n noduri (fiecare element corespunde unui nod)
while "există cel puțin două clustere"
Determină cele mai apropiate două clustere (C_1 și C_2)
Reunește C_1 și C_2 într-un nou cluster C
Calculează distanța dintre C și toate celelalte clustere
Adaugă în arbore un nod corespunzător lui C și unește-l cu nodurile C_1 și C_2
Elimină din D liniile și coloanele corespunzătoare nodurilor C_1 si C_2
Adăugă în D o linie și o coloană corespunzătoare clusterului C
endwhile
return ***A***

Metode ierarhice

Observații:

- In funcție de măsura utilizată pentru a calcula disimilaritatea dintre clustere (single link, complete link sau average link) există mai multe variante de algoritmi

- Există și variante care reunesc mai multe clustere la o etapă (în acest caz se reunesc toate clusterele pentru care disimilaritatea dintre ele este mai mică decât un prag dat și nu doar cele pentru care este atinsă valoarea minimă). In acest caz arborele obținut nu va mai fi neapărat un arbore binar

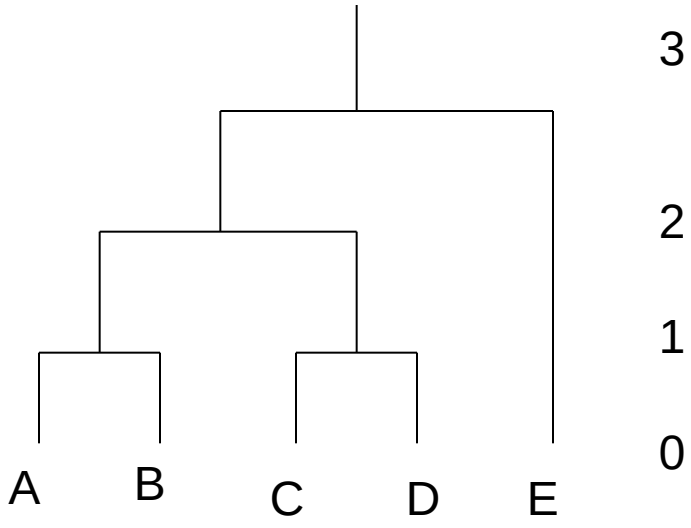
- Algoritmii ierarhici sunt de complexitate cel puțin pătratică în raport cu numărul de date

Metode ierarhice

Observatii:

- dendrogramele pot fi reprezentate prin structuri simple în care fiecărui nivel îi corespunde setul de clustere identificate

Exemplu:



Descriere dendrogramă:

{(0,5,{{A},{B},{C},{D},{E}}),
(1,3,{{A,B},{C,D},{E}}),
(2,2,{{A,B,C,D},{E}}),
(3,1,{A,B,C,D,E})}

(nivel, număr clustere, structura)

Metode ierarhice

Observații:

- Metodele ierarhice sunt similare cu tehnicile de identificare a arborelui minim de acoperire sau cu identificarea clicilor într-un graf asociat datelor:
  - nodurile corespund datelor iar muchiile corespund perechilor de pentru care disimilaritatea este mai mică decât un prag dat;
  - costul asociat unei muchii este proporțional cu valoarea disimilarității dintre noduri.

- In cazul variantelor divizive se pornește de la arborele minim de acoperire și se elimină succesiv muchii în ordinea descrescătoare a costului lor

Metode partitionale

Particularități:

- Generează o singură partiție
- Fiecărui cluster i se poate asocia un reprezentant (ex: centroid)
- Elementele clasei se stabilesc în funcție de distanța față de reprezentantul clasei (criteriul distanței minime)
- Avantaj: In general sunt mai eficiente decât metodele ierarhice
- Dezavantaj: Necesită cunoașterea numărului de clase

Metode partitionale

Principiu general:

In cazul a k clustere C_1, C_2,… ,C_k se identifică reprezentanții (centroizii) R_1, R_2,… ,R_k care minimizează distanța față de date

$$E(R_1, R_2, ..., R_K) = \sum_{k=1}^{K} \sum_{x \in C_k} d(R_k, x)^2$$

Observații:

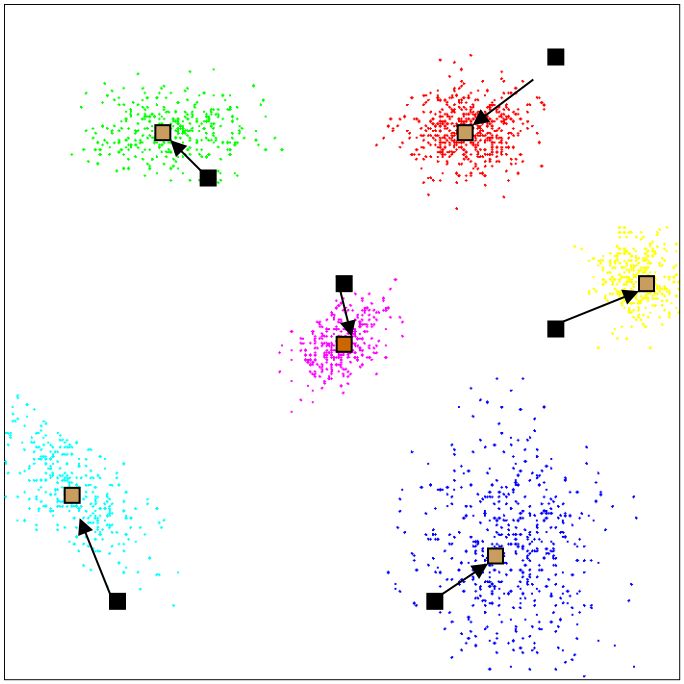
1. Datele sunt asignate la clase pe criteriul distanței minime

2. Identificarea reprezentanților este echivalentă cu rezolvarea unei probleme de optimizare

Metode partitionale

Algoritmul mediilor (K-means):

- Reprezentanții sunt medii aritmetice ale elementelor clusterului
- Se pornește cu reprezentanți initializați aleator (selectați aleator din setul de date)
- Proces iterativ (alg. Lloyd):
  - Se asignează datele la clase folosind criteriul distanței minime (sau a celui mai apropiat reprezentant)
  - Se recalculează reprezentanții ca fiind medii ale elementelor asignate fiecărei clase



Metode partitionale

K-means:

- $R^k := X^{q(k)}$, k=1..K; q(1),q(2),…,q(K) sunt generate aleator în mulțimea de indici ai datelor astfel încât să fie distincte
- **repeat**
  - **for** fiecare dată X^i
    Determină q(i) astfel încât $d(X,R^{q(i)}) <= d(X,R^k)$, pt orice k
    Asignează X clasei q(i)
  - **for** k=1:K
    $R^{k:}$ = media elementelor ce au fost asignate clasei k

  until "nu s-au mai efectuat modificari ale reprezentanților"

Obs:

- Reprezentanții nu sunt neapărat vectori din setul de date
- Ordin de complexitate liniar în raport cu numărul de date prelucrate (dar depinde de numărul de iterații)

Metode partitionale

- Algoritmul k-means (varianta Lloyd) este eficient dar în fiecare etapă se bazează pe mutarea mai multor date dintr-un cluster în altul
- O variantă mai conservativă ar fi să se mute o singură dată și anume cea care asigură cea mai mare descreștere a costului partiționării
  - Cu cât costul partiționării este mai mic cu atât este mai bună partiționarea
  - Costul partiționării poate fi calculat în diferite moduri

Metode partitionale

- Costul unei partiționări poate fi specificat prin:

$$\text{cost}(P) = \sum_{i=1}^{n} d(x_i, R)^2$$

$$\text{cost}(P) = \sum_{i=1}^{n} d(x_i, R)$$

$$\text{cost}(P) = \max_{i=\overline{1,n}} d(x_i, R)$$

Obs:

1. d(x,R) reprezintă distanța de la x la cel mai apropiat reprezentant din setul $R=\{R^1,\ldots,R^k\}$
2. Alegând R care minimizează prima variantă de cost se ajunge la reprezentanți de tip centroid iar în al treilea caz se ajunge la reprezentanți de tip medoid

Metode partitionale

O variantă de tip greedy a algoritmului K-means (modificare minimală cu câștig maximal)

ProgressiveGreedyK-Means(*k*)
Determină o partiție inițială aleatoare
repeat
  bestChange := 0
  for fiecare cluster *C*
    for fiecare element *i* care nu aparține lui *C*
     II daca prin mutarea lui *i* in clusterul *C* se reduce Cost(P)
       if cost(*P*) – cost($P_{i \to C}$) > *bestChange*
         bestChange := cost(*P*) – cost($P_{i \to C}$)
         i^*:=*i*
         C^* :=*C*
  if *bestChange* > 0
    schimba partiția *P* mutând i^* în C^*
  until bestChange=0
  return *P*

Alti algoritmi partitionali (variante ale algoritmului kMeans)

ISODATA (Iterative Self-Organizing Data Analysis)

- Permite estimarea numărului adecvat de clusteri
- Se bazează pe următorii parametri setați de către utilizator:
  - Numărul minim de date într-un cluster (Nmin)
  - Distanța minimă dintre două clustere (Dmin)
  - Varianța maximă din cadrul unui cluster (Vmax)

Alti algoritmi partitionali (variante ale algoritmului kMeans)

ISODATA (Iterative Self-Organizing Data Analysis)

Se bazeaza pe operații de fuziune respectiv divizare a clusterilor:

- Dacă numărul de date dintr-un cluster este mai mic decât Nmin atunci clusterul fuzionează cu cel mai apropiat alt cluster

- Dacă distanța dintre două clustere (ex: distanța dintre reprezentanți) este mai mică decat Dmin atunci clusterele fuzionează

- Dacă varianța unui cluster k depășește Vmax și numărul de date pe care le conține depășește 2*Nmin atunci clusterul este divizat în două clustere: se determină componenta j pentru care varianța e maximă și din reprezentantul R_k se construiesc doi reprezentanți R' si R'' înlocuind componenta j din R_k cu $R_k(j)$-d respectiv $R_k(j)$+d (d este un parametru specificat de către utilizator)

Alti algoritmi partitionali (variante ale algoritmului kMeans):

Fuzzy Cmeans - se presupune că o dată nu aparține exclusiv unui cluster

- Construiește o matrice de dimensiune NxK (N=număr date, K=număr clustere) în care pe linia i coloana j se află o valoare din [0,1] care exprimă gradul de apartenență al datei i la clusterul k
- Datele nu sunt asignate în mod rigid la clusteri (“soft clustering” in loc de “hard clustering”) iar decizia asignării la un cluster se poate lua pe baza gradului de apartenență determinat prin ajustare iterativă
- La fel ca în cazul algoritmului kMeans numărul de clustere (K) trebuie specificat de la început

Alti algoritmi partitionali (variante ale algoritmului kMeans):

Algoritm Fuzzy C-means [Bezdek, 1981]

- Inițializarea matricii cu valori de apartenență (U)
- **do**
  - Calculează centroizii(V_{1j})
  - Ajustează valorile de apartenență (u_{ij})

  while <există modificări semnificative în valorile din matricea de apartenență>
- Asignează datele la clasa pentru care valoarea de apartenență este maximă

Calcul centroizi

$$v_j = \frac{\sum_{i=1}^{n} u_{ij}^m x_i}{\sum_{i=1}^{n} u_{ij}^m}, \quad j = \overline{1,c}$$

m>1 este un parametru (ex. m=2)

Estimarea valorilor de apartenență

$$u_{ij} = \frac{1}{\left\|x_i - v_j\right\|^{2/(m-1)} \sum_{k=1}^{c} 1/\left\|x_i - v_k\right\|^{2/(m-1)}}$$

$$i = \overline{1,n},\ j = \overline{1,c}$$

Biclustering

Specificul datelor:

- Sunt măsurători ale nivelului de expresie genică (concentrație relativă de ARN mesager) pentru diferite gene în diferite condiții experimentale/provenind de la diferite țesuturi
- Datele sunt grupate într-o matrice in care:
  - Fiecare linie reprezintă valori asociate unei gene
  - Fiecare coloană reprezintă valori asociate unei condiții experimentale

Biclustering

Specificul datelor:

- Scopul urmărit este identificarea grupurilor de gene care se comportă “similar” în anumite condiții experimentale (este puțin plauzibil ca gene diferite să se comporte similar în toate condițiile experimentale)
- Gruparea presupune identificarea atât a genelor similare cât și a condițiilor experimentale în care se manifestă similaritatea => clustering atât la nivelul liniilor cât și la nivelul coloanelor
- Acest tip de grupare este denumită biclustering (sau co-clustering)

Biclustering

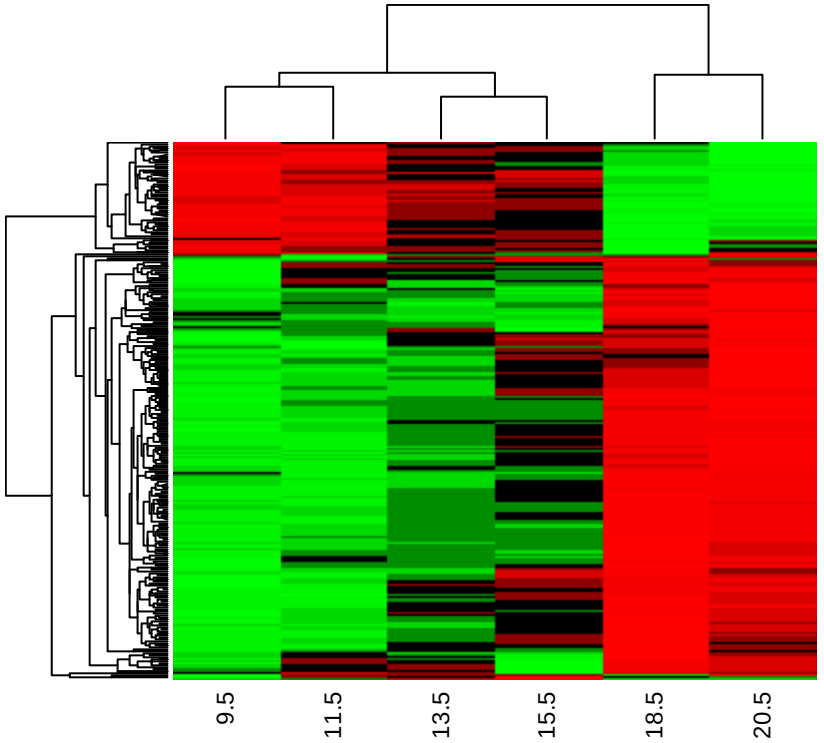
Exemplu

Date: drojdia de bere (Matlab)

Functie Matlab: clustergram

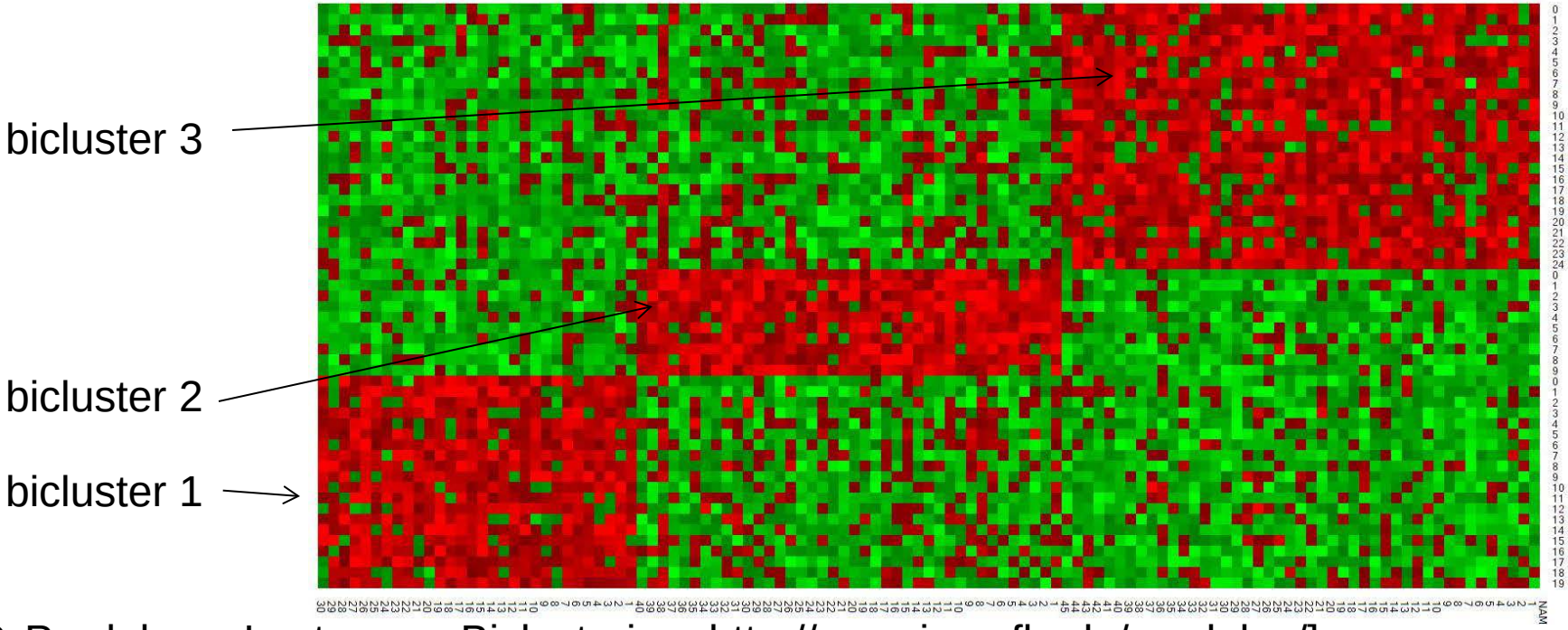
Legenda:

- roșu: valori peste medie
- negru: valori egale cu media
- verde: valori sub medie



Biclustering

- Algoritmii de biclustering au fost proiectați pentru a realiza gruparea simultană atât la nivelul datelor (genelor) cât și la nivelul atributelor (condițiilor experimentale) => identificarea în matricea datelor a unor submatrici (nu neapărat disjuncte)
- Submatricile sunt denumite biclustere



[P. Pardalos – Lecture on Biclustering, http://www.ise.ufl.edu/pardalos/]

Biclustering

Direct clustering (Hartigan, 1972)

- Se pornește de la matricea de date văzută ca un singur bloc după care se identifică linia și coloana unde să se realizeze partiționarea; identificarea se bazează pe minimizarea variației datelor în fiecare bloc; procesul se repetă pentru fiecare bloc identificat

- Inițial nu a fost concepută pentru analiza expresiei genice ci pentru analiza datelor electorale

Biclustering

Metoda Cheng-Church (2000)

Idee:

- se caută succesiv câte un bicluster utilizând un criteriu de optim specific (vezi slide următor)
- după identificarea unui bicluster elementele corespunzătoare porțiunii pe care o ocupă în matrice sunt înlocuite cu valori aleatoare
- se continuă procesul cu identificarea unui nou bicluster

Biclustering

Metoda Cheng-Church (2000)

Identificarea unui bicluster = problemă de optimizare cu restricții

- Pentru o matrice A[1..m,1..n] conținând valori ale expresiei genice (în cazul a m gene și a n condiții) se caută un bicluster, adică o submatrice A[I,J] unde I este submulțime a lui {1,...,m} iar J este submulțime a lui {1,...,n} astfel încât:
  - Biclusterul să fie cât mai extins (I și J sa fie submulțimi cât mai mari)
  - Variabilitatea datelor în cadrul biclusterului să fie mai mică decât un prag dat

- Cheng si Church au propus o măsură specifică a variabilității unui bicluster (I,J) numită *reziduu pătratic mediu* (mean squared residue score)

Biclustering

Metoda Cheng-Church (2000)

Reziduu pătratic mediu:

$$H(I,J) = \frac{1}{card(I)\cdot card(J)} \sum_{i\in I, j\in J} (a_{ij} - a_{iJ} - a_{Ij} + a_{IJ})^2$$

$$a_{iJ} = \frac{1}{card(J)} \sum_{j\in J} a_{ij} \qquad a_{Ij} = \frac{1}{card(I)} \sum_{i\in I} a_{ij}$$

$$a_{IJ} = \frac{1}{card(I)\cdot card(J)} \sum_{i\in I, j\in J} a_{ij}$$

Observatie:

H(I,J)=0 indică faptul ca nivelele de exprimare ale genelor fluctuează la unison (cazul în care elementele biclusterului sunt identice este doar un caz particular)

Exemplu de matrice cu H=0:

1 2 3	2
3 4 5	4
5 6 7	6
3 4 5	

Biclustering

Varianta bazată pe tehnica forței brute

Intrare: matricea datelor A[1..m,1..n]
parametrul δ (pragul maxim pentru H(I,J))
Iesire: δ-bicluster = pereche (I,J) pentru care H(I,J) < δ cu I si J maximale

Structura generală a unui algoritm bazat pe tehnica forței brute:

I={1,...,m} J={1,....,n}
repeat
- calculează H pentru fiecare eliminare/inserare de linie/coloană
- aplică operația pentru care descreșterea lui H este cea mai mare

until <nu există operație care să micșoreze pe H> **or** <H a devenit mai mic decat δ>

Complexitate: O((m+n)mn)

Biclustering

Algoritmul Cheng-Church

Etapa 0: inițializare I={1,...,m}, J={1,...,n}; setare valoare α

Etapa 1: eliminare multiplă linii/coloane

- Se elimină liniile i cu proprietatea $\frac{1}{card(J)}\sum_{j\in J}(a_{ij}-a_{iJ}-a_{Ij}+a_{IJ})^2 > \alpha H(I,J)$
- Se elimină coloanele j cu propr. $\frac{1}{card(I)}\sum_{i\in I}(a_{ij}-a_{iJ}-a_{Ij}+a_{IJ})^2 > \alpha H(I,J)$

Etapa 2: Se elimină secvențial linia i, coloana j pentru care d(i) respectiv d(j) sunt maxime

$$d(i)=\sum_{i\in I}(a_{ij}-a_{iJ}-a_{Ij}+a_{IJ})^2/card(J)$$

$$d(j)=\sum_{i\in I}(a_{ij}-a_{iJ}-a_{Ij}+a_{IJ})^2/card(I)$$

Etapa 3: Se adaugă liniile și coloanele care nu măresc valoarea reziduului

Ordin complexitate: O(mn)

Alte metode

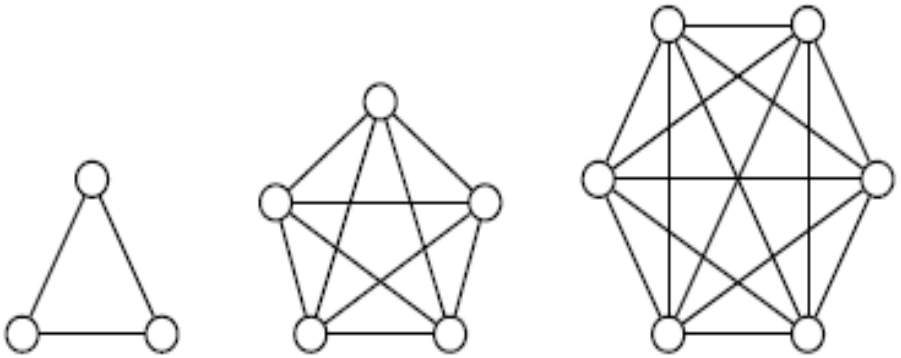
- Algoritmi de grupare bazați pe grafuri
- Algoritmi de grupare pentru volume mari de date

Algoritmi de grupare bazati pe grafuri

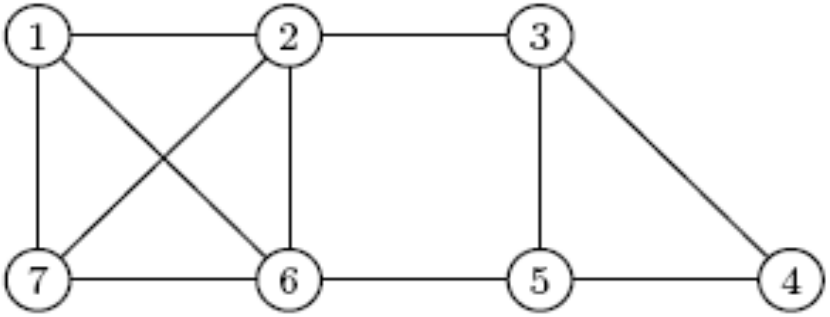
- Pornind de la un set de n date se poate construi un graf caracterizat prin:
  - Mulțimea nodurilor coincide cu mulțimea datelor
  - Nodurile sunt unite între ele prin muchii etichetate cu valoarea măsurii de disimilaritate dintre datele asociate
- Gruparea datelor este echivalentă cu identificarea unui subgraf constituit din componente conexe cu conectivitate totală (numite clici)
- Datele din fiecare clică trebuie să fie suficient de similare între ele (deci costurile muchiilor corespunzătoare ar trebui să fie mai mici decât un anumit prag)

Algoritmi de grupare bazati pe grafuri

- Exemple de clici:

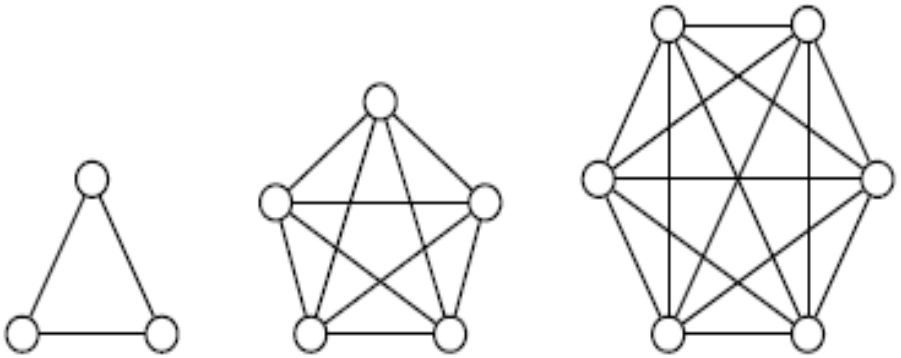


- Determinarea clicilor se realizează prin eliminarea unor muchii din graful curent (cele al căror cost depășește pragul)

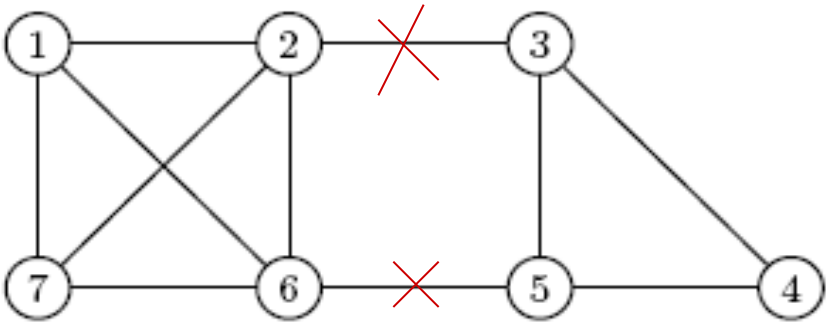


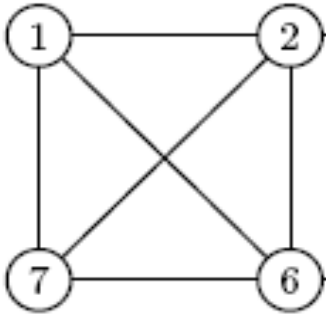
Algoritmi de grupare bazati pe grafuri

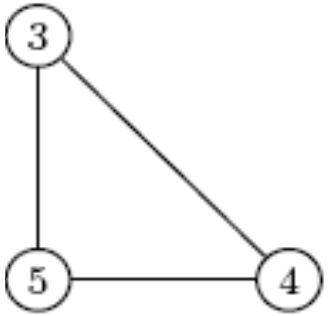
- Exemple de clici:



- Determinarea clicilor se realizeaza prin eliminarea unor muchii din graful curent (cele al caror cost depaseste pragul)







Algoritmi de grupare bazati pe grafuri

- Problema care trebuie rezolvată:
  - pornind de la un graf să se determine (printr-un număr cât mai mic de eliminări de muchii) un graf constituit din componente conexe de tip clică
- Este o problema din categoria celor NP-dificile
- Metodă euristică de rezolvare:
  - **CAST** (Cluster Affinity Search Technique)
  - Metoda se bazează pe calculul distanței dintre o dată și un cluster definită ca fiind media aritmetică a distanțelor dintre data respectivă și toate elementele clusterului
  - Apropierea sau îndepărtarea față de un cluster se definește în funcție de un prag T:
    - Daca d(x,C) <= T atunci x se consideră apropiată de clusterul C
    - Daca d(x,C) > T atunci x se consideră îndepartată de clusterul C

Algoritmi de grupare bazati pe grafuri

1. CAST(*S*, *G*, T)
2. ***P*** ← Ø
3. **while *S*** ≠ Ø
4. *v* ← nod de grad maxim in graful ***G***
5. ***C*** ← {*v*}
6. **while** "exista o data x care fie este apropiată de **C** dar nu face parte din **C** fie este îndepartată de **C** dar face parte din **C**"
7. Determină data cea mai apropiata de **C** care nu face parte din **C**
8. și o adaugă la **C**
9. Elimină data din **C** pentru care distanța față de **C** este maximă
10. **endwhile**
11. Adauga clusterul ***C*** la partiția ***P***
12. ***S*** ← ***S*** \ ***C***
13. Elimină nodurile corespunzatoare din graful G
14. **endwhile**
15. return ***P***

Notații:

S – setul de date

G – graful asociat datelor corespunzator pragului T (doar nodurile corespunzatoare unor date aflate la o distanta mai mica decat T sunt unite prin muchii)

P - partitia

Algoritmi de grupare bazati pe grafuri

Observatii:

- algoritmul nu garantează obținerea partiționării corespunzătoare numărului minim de muchii eliminate

- în schimb, ca majoritatea euristicilor, este eficient

Algoritmi pentru seturi mari de date

Metodele prezentate până acum se caracterizează prin:

- necesită încărcarea în memorie a întregului set de date

- sunt fie de complexitate $O(n^2)$ fie necesită parcurgerea de mai multe ori a setului de date

Deci nu sunt adecvate pentru:

- seturi de date de dimensiuni mari
- date dinamice (baze de date care își modifică conținutul pe parcurs)

Algoritmi pentru seturi mari de date

Un algoritm adecvat pentru prelucrarea seturilor mari de date ar trebui sa aibă următoarele caracteristici:

- sa parcurgă o singură dată baza de date

- sa poată fi întrerupt în timpul execuției și reluat ulterior

- sa aibă caracter incremental (să permită actualizarea rezultatului procesului de clusterizare ca urmare a adăugării sau ștergerii de elemente din baza de date)

- să nu necesite încărcarea în memorie a întregului set de date

- sa permită parcurgerea în diferite moduri a bazei de date (inclusiv selecție aleatoare a elementelor)

Algoritmi pentru seturi mari de date

Idee de bază:

- se încarcă la un moment dat în memorie un subset de date
- se partiționează fiecare astfel de subset
- se "combină" partiționările astfel obținute

Obs.: ideea este utilizată și în cazul prelucrării datelor distribuite în baze de date aflate în locații diferite; există variante distribuite pentru algoritmul kMeans și pentru algoritmii ierarhici

Exemple de algoritmi:

1. BIRCH (Balanced Iterative Reducing and Clustering using Hierarchies)
2. CURE (Clustering Using REpresentatives)

Algoritmul BIRCH

Caracteristici generale:

- presupune că resursele de memorie internă sunt limitate
- parcurge o singură dată baza de date
- este incremental (permite asimilarea de noi date)
- se bazează pe principiul grupării ierarhice
- folosește o tehnică de tratare a excepțiilor (outlier data)
- poate fi aplicat doar pentru date numerice

Ideea de bază:

- construiește un arbore în care se capturează informațiile necesare pentru clusterizare

- utilizează informațiile din arbore pentru asignarea datelor la clusterele identificate

Algoritmul BIRCH

Noțiuni folosite:

1. Clustering Feature (CF):

un triplet (N,LS,SS) care se asociază unui cluster si conține:

N=numărul de date din cluster

M=suma datelor din cluster (pe baza lui și a lui N se poate determina centroidul clusterului)

S=suma pătratelor datelor din cluster (pe baza lui se poate calcula varianța datelor din cluster) – pătratul se calculează separat pentru fiecare componentă

$$Var(X) = \frac{S}{N} - \left(\frac{M}{N}\right)^2$$

Obs: in aceasta relație se presupune că S și M sunt valori scalare corespunzătoare uneia dintre componentele vectorilor din set

Algoritmul BIRCH

Noțiuni folosite:

2. CF Tree: arbore echilibrat caracterizat prin:

- factor de ramificare (nr. de fii pe care îi poate avea un nod limitat – B)

- fiecare nod intern reprezintă un cluster și conține câte un CF pentru fiecare dintre fii săi (aceștia corespund unor subclustere)

- fiecare nod frunză reprezintă si el un cluster si conține câte un CF pt. fiecare subcluster (diametrul unui subcluster corespunzator unui nod frunză trebuie să fie mai mic decât un prag prestabilit, T)

Algoritmul BIRCH

Etape:

1. construiește arborele (CF Tree) pornind de la date – frontiera va conține partiția finală

2. reorganizează arborele aplicând un algoritm de grupare (ex: alg ierarhic aglomerativ) asupra subclusterilor din nodurile frunză

3. asignează datele la clustere pe criteriul distanței minime față de centroid (datele care se afla la o distanță mare față de toți centroizii sunt considerate excepții și sunt ignorate)

Algoritmul BIRCH

Construire arbore:

for fiecare x din setul de date
determină nodul frunză unde trebuie inserat
if (condiția de prag nu e încalcată)
then adaugă x la clusterul corespunzator și actualizează toate CF-urile corespunzătoare pe ramură în sus până la rădăcină
else if există spațiu pentru inserarea lui x (noua intrare în nod)
then inserează x ca un cluster constituit dintr-un singur element
else divizează nodul frunza și redistribuie CF-urile
endif
endif
endfor

Algoritmul CURE

Caracteristici generale:

- permite tratarea excepțiilor

- pornește de la prelucrarea unui eșantion (un subset din date)

- are atât o componenta ierarhică cât și una partițională

- folosește pentru fiecare cluster mai mulți reprezentanți (în felul acesta pot fi identificate clustere cu forma arbitrară)

Algoritmul CURE

Etape:

1. se determină un eșantion din date

2. se partiționează eșantionul în p partiții

3. se aplică un algoritm ierarhic de grupare pentru datele din fiecare partiție; în felul acesta se obține o prima aproximare a grupării

4. se elimină excepțiile (clusterele mici)

5. se clusterizează toate datele din eșantion folosind reprezentanții clusterelor detectate la etapa 3

6. se asignează la clustere datele din întregul set (folosind criteriul distanței minime față de reprezentanții clusterelor)

Algoritmul CURE

Algoritmul de la etapa 5:

```
T=build(D)   // arbore k-D
Q=heapify(D) // Q=heap cu clusterele in ordinea crescatoare a
                distantelor dintre ele
repeat
   u=min(Q)
   delete(Q,u.closest)
   w=merge(u,v); delete(T,u); delete(T,v); insert(T,w)
   for x din Q
        x.closest = cel mai apropiat cluster de x
         if x este cel mai apropiat de w then w.closest=x
         endif
   insert(Q,w)
 until numarul de noduri din Q este k
```