M1 Info - Systemes Complexes Avances

Cours 5 - Algorithmes de detection de communautes

Girvan-Newman, Louvain, Leiden

Semestre Automne 2022-2023 - Université Côte D'azur

Christophe Crespelle
christophe.crespelle@univ-cotedazur.fr

UNIVERSITÉ CÔTE D'AZUR

DIGITAL SYSTEMS FOR HUMANS
GRADUATE SCHOOL AND RESEARCH

Communities in complex networks

What is a community ?

"Moral" definition

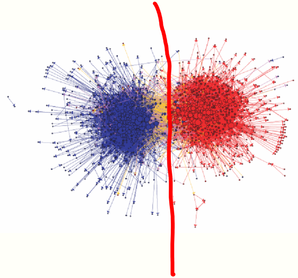
- A group of nodes that share something...
  - People with a common interest
  - Web pages with similar content
  - Proteins realising a common function

Communities in complex networks

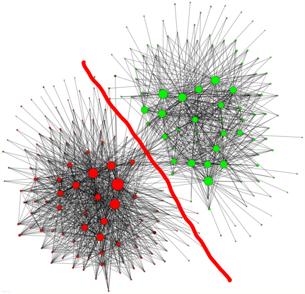
What is a community ?

"Moral" definition

- A group of nodes that share something...
  - People with a common interest
  - Web pages with similar content
  - Proteins realising a common function
- ... that makes them be in relationship in the network !



Political blogs in US



Languages in Belgium

Communities in complex networks

What is a community ?

Structural definition

- A highly connected group of nodes

Communities in complex networks

What is a community ?

Structural definition

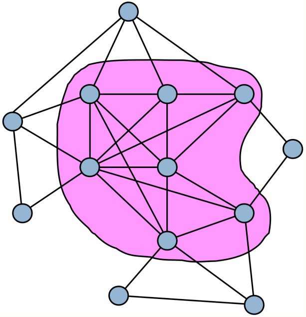
- A highly connected group of nodes
  - Density inside the community much higher than global density of the network

Communities in complex networks

What is a community?

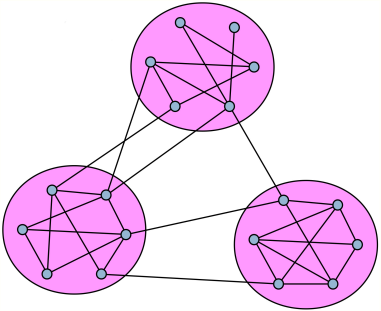
Structural definition

- A highly connected group of nodes
  - Density inside the community much higher than global density of the network
  - Only few edges toward the rest of the network



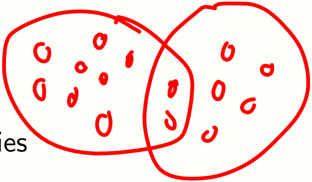
Types of structural communities

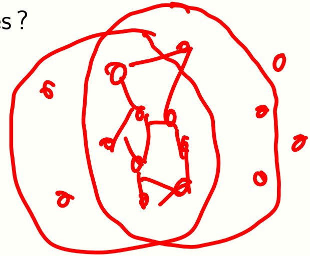
- Partition of the nodes into dense parts sparsely connected between them
  - High density inside communities
  - Few edges between communities

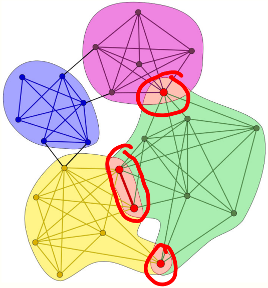


Types of structural communities

- Partition of the nodes into dense parts sparsely connected between them
  - High density inside communities
  - Few edges between communities
- Overlapping communities
  A node can belong to several communities
  - more realistic
  - problem : how to separate communities ?

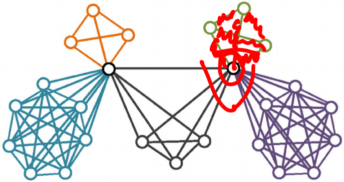






Types of structural communities

- Partition of the nodes into dense parts sparsely connected between them
  - High density inside communities
  - Few edges between communities
- Overlapping communities
  A node can belong to several communities
  - more realistic
  - problem : how to separate communities ?
- Partition of the links
  - a link belong to exactly one community
  - a node can have links in different communities



Partition of the nodes

Various approaches, among them :

- random walks
- spectral methods
- hierarchical clustering
- divisive methods
- Louvain, Leiden

Partition of the nodes

Various approaches, among them :

- random walks
- spectral methods
- hierarchical clustering
- **divisive methods**
- **Louvain, Leiden**

Partition of the nodes

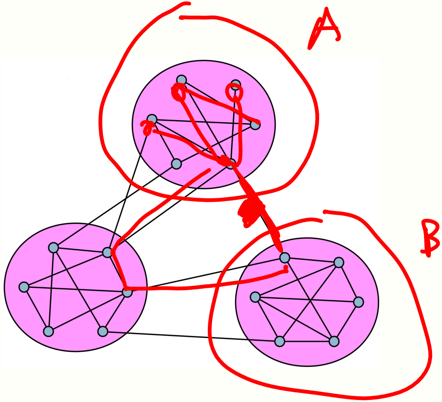
Various approaches, among them :

- random walks
- spectral methods
- hierarchical clustering
- **divisive methods**
- **Louvain, Leiden**

Divisive approach : Girvan & Newman 2002

The idea :

1. identify inter-community links
2. remove them



How to identify inter-community links ?

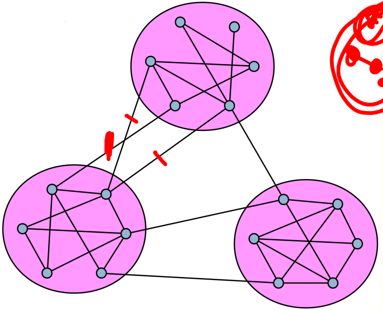
- Betweenness centrality of links
  - $C_B(e) = \sum_{s \neq t} \frac{\sigma_{st}(e)}{\sigma_{st}}$ where 0 ou 1
    - $\sigma_{st} = \#$ shortest paths from s to t
    - $\sigma_{st}(e) = \#$ shortest paths from s to t containing e

How to identify inter-community links ?

- Betweenness centrality of links
  - $C_B(e) = \sum_{s \neq t} \frac{\sigma_{st}(e)}{\sigma_{st}}$ where
    - $\sigma_{st} = \#$ shortest paths from s to t
    - $\sigma_{st}(e) = \#$ shortest paths from s to t containing e
  - high betweenness $\Leftrightarrow$ e is on a high proportion of shortest paths for a high proportion of pairs of nodes

How to identify inter-community links?

- Betweenness centrality of links
  - $C_B(e) = \sum_{s \neq t} \frac{\sigma_{st}(e)}{\sigma_{st}}$ where
    - $\sigma_{st} = \#$ shortest paths from s to t
    - $\sigma_{st}(e) = \#$ shortest paths from s to t containing e
  - high betweenness $\Leftrightarrow$ e is on a high proportion of shortest paths for a high proportion of pairs of nodes



The algorithm (?)

- Algo Girvan-Newman(G)
    1. Compute the betweenness centrality of all links e of G

The algorithm (?)

- Algo Girvan-Newman(G)
  1. Compute the betweenness centrality of all links e of G
  2. for all links e in decreasing betweenness centrality do

The algorithm (?)

- Algo Girvan-Newman(G)
  1. Compute the betweenness centrality of all links e of G
  2. for all links e in decreasing betweenness centrality do
     - remove e from G

The algorithm (?)

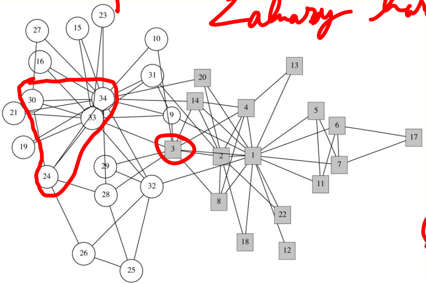
- Algo Girvan-Newman(G)
    1. Compute the betweenness centrality of all links e of G
    2. for all links e in decreasing betweenness centrality do
        - remove e from G
        - update the connected components of G

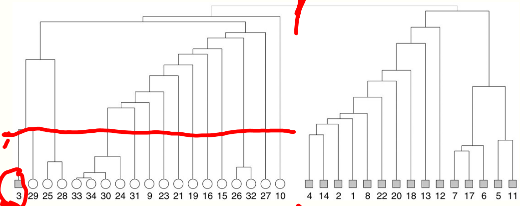
The algorithm (?)

- Algo Girvan-Newman(G)
  1. Compute the betweenness centrality of all links e of G
  2. for all links e in decreasing betweenness centrality do
     - remove e from G
     - update the connected components of G
  3. output the dendogram of G

The algorithm (?)

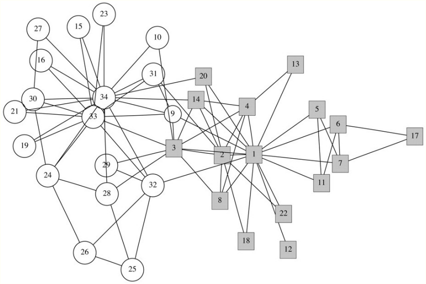
- Algo Girvan-Newman(G)
  1. Compute the betweenness centrality of all links e of G
  2. for all links e in decreasing betweenness centrality do
     - remove e from G
     - update the connected components of G
  3. output the dendogram of G

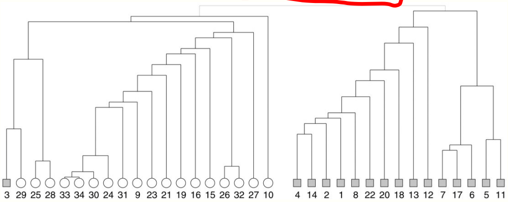




The algorithm

- Algo Girvan-Newman(G)
  1. Compute the betweenness centrality of all links e of G
  2. for all links e in decreasing betweenness centrality do
     - remove e from G
     - update the connected components of G
     - **update the betweenness centrality of all links**
  3. output the dendogram of G





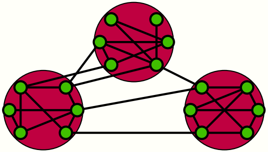
The algorithm

- Algo Girvan-Newman(G)
  1. Compute the betweenness centrality of all links e of G
  2. for all links e in decreasing betweenness centrality do
     - remove e from G
     - update the connected components of G
     - update the betweenness centrality of all links
  3. output the dendogram of G

- Complexity
  - betweenness for all links : $O(nm)$
  - connected components : $O(m)$
  - m iterations
  - Overall : $O(nm^2)$

The Louvain algorithm

- Idea : optimize a quality function for node partitions



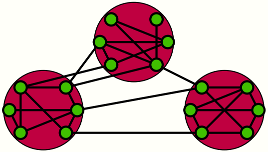
m

- **modularity** :maximize(#edges inside - #edges outisde)
  ⇔ maximize(#edges inside)

m - #int

The Louvain algorithm

- Idea : optimize a quality function for node partitions



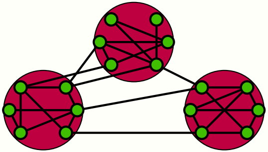
  - **modularity** :maximize(#edges inside - #edges outisde)
    $\Leftrightarrow$ maximize(#edges inside)

- Problem... the best partition is a single community ! ! !

The Louvain algorithm

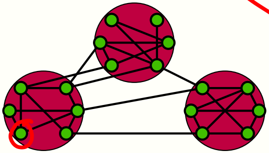
- Idea : optimize a quality function for node partitions

1 - 1 = 0

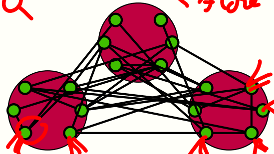


  - **modularity** :maximize(#edges inside - #edges outisde)
    $\Leftrightarrow$ maximize(#edges inside)

- Problem... the best partition is a single community ! ! !
- Correction : compare to a randomized version of the network

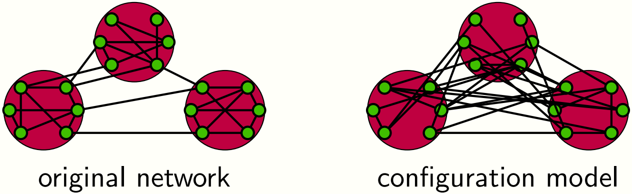


original network



configuration model

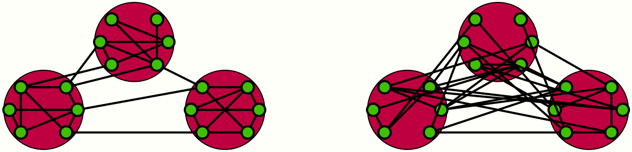
Modularity



- Proportion of edges inside communities

A the adjacency matrix of G
k_i the degree of node i
c_i the community of node i
δ is the Kronecker symbol : $\delta(c_i, c_j) = 1$ iff $c_i = c_j$

Modularity



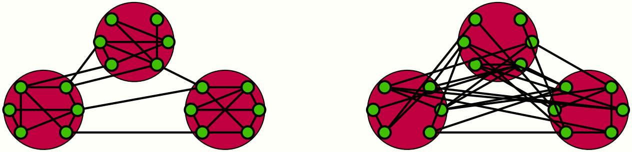
original network   configuration model

- Proportion of edges inside communities

  A the adjacency matrix of G
  k_i the degree of node i
  c_i the community of node i
  δ is the Kronecker symbol : $\delta(c_i, c_j) = 1$ iff $c_i = c_j$ et 0 sinon.

  - In the original network : $\frac{1}{2m}\sum_{i,j\in V} A_{ij}\delta(c_i, c_j)$ where

Modularity

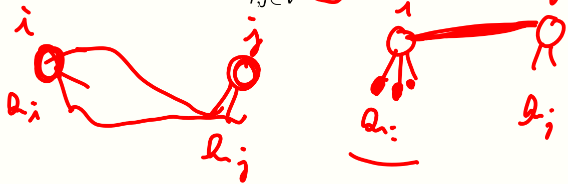


original network — configuration model

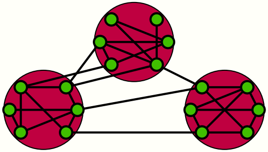
- Proportion of edges inside communities

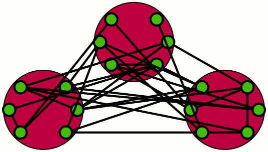
  A the adjacency matrix of G
  k_i the degree of node i
  c_i the community of node i
  δ is the Kronecker symbol : $\delta(c_i, c_j) = 1$ iff $c_i = c_j$

  - In the original network : $\frac{1}{2m} \sum_{i,j \in V} A_{ij} \delta(c_i, c_j)$ where
  - In the configuration model : $\frac{1}{2m} \sum_{i,j \in V} \frac{k_i k_j}{2m} \delta(c_i, c_j)$



Modularity


original network


configuration model

- Proportion of edges inside communities

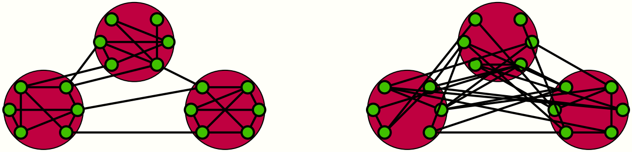
  A the adjacency matrix of G
  k_i the degree of node i
  c_i the community of node i
  δ is the Kronecker symbol : $\delta(c_i, c_j) = 1$ iff $c_i = c_j$

  - In the original network : $\frac{1}{2m} \sum_{i,j \in V} A_{ij} \delta(c_i, c_j)$ where
  - In the configuration model : $\frac{1}{2m} \sum_{i,j \in V} \frac{k_i k_j}{2m} \delta(c_i, c_j)$

- **modularity** : $Q(\mathcal{P}) = \frac{1}{2m} \sum_{i,j \in V} [A_{ij} - \frac{k_i k_j}{2m}] \delta(c_i, c_j)$

  $\in [-1,1]$ $= \frac{1}{2m} \sum_{c \in \mathcal{P}} [e_c - \frac{a_c^2}{2m}]$

Modularity



original network configuration model

- Proportion of edges inside communities

  A the adjacency matrix of G
  k_i the degree of node i
  c_i the community of node i
  δ is the Kronecker symbol : $\delta(c_i, c_j) = 1$ iff $c_i = c_j$

  - In the original network : $\frac{1}{2m} \sum_{i,j \in V} A_{ij} \delta(c_i, c_j)$ where
  - In the configuration model : $\frac{1}{2m} \sum_{i,j \in V} \frac{k_i k_j}{2m} \delta(c_i, c_j)$

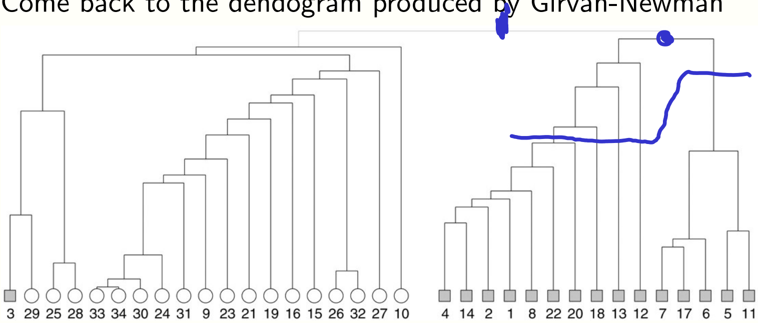
- **modularity** :

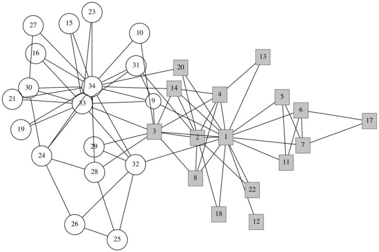
$$Q(\mathcal{P}) = \frac{1}{2m} \sum_{i,j \in V} [A_{ij} - \frac{k_i k_j}{2m}] \delta(c_i, c_j) = \frac{1}{2m} \sum_{c \in \mathcal{P}} [e_c - \frac{a_c^2}{2m}]$$

  - **NP-hard** to maximize modularity

Utility of modularity

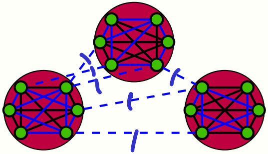
- Come back to the dendogram produced by Girvan-Newman



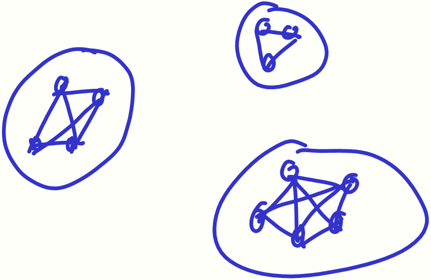


Other quality functions

- Distance to cluster graphs

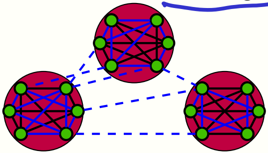


  - dist-cluster($\mathcal{P}$)=#missing edges inside + #edges outside



Other quality functions

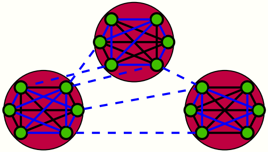
- Distance to cluster graphs



  - dist-cluster($\mathcal{P}$)=#missing edges inside + #edges outside
  - **NP-hard** to minimize distance to cluster graphs

Other quality functions

- Distance to cluster graphs

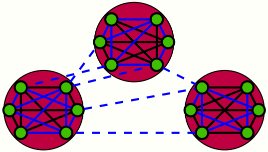


  - dist-cluster($\mathcal{P}$)=#missing edges inside + #edges outside
  - **NP-hard** to minimize distance to cluster graphs

- Constant Potts Model
  - CPM($\mathcal{P}$)=$\sum_c [e_c - \gamma\binom{n_c}{2}]$
    where e_c=# edges inside communauty c
    and n_c=# nodes in communauty c
    γ is a chosen constant ≤ 1

Other quality functions

- Distance to cluster graphs

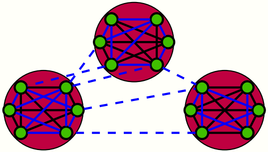


  - dist-cluster($\mathcal{P}$)=#missing edges inside + #edges outside
  - **NP-hard** to minimize distance to cluster graphs

- Constant Potts Model
  - CPM($\mathcal{P}$)=$\sum_c [e_c - \gamma \binom{n_c}{2}]$
    where e_c=# edges inside communauty c
    and n_c=# nodes in communauty c
    γ is a chosen constant ≤ 1
  - for $\gamma = 0$?

Other quality functions

- Distance to cluster graphs

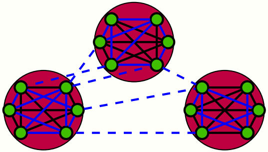


  - dist-cluster($\mathcal{P}$)=#missing edges inside + #edges outside
  - **NP-hard** to minimize distance to cluster graphs

- Constant Potts Model
  - CPM($\mathcal{P}$)=$\sum_{c}[e_c - \gamma\binom{n_c}{2}]$
    where e_c=# edges inside communauty c
    and n_c=# nodes in communauty c
    γ is a chosen constant ≤ 1
  - for $\gamma = 0$?
  - for $\gamma = 1$?

Other quality functions

- Distance to cluster graphs



  - dist-cluster($\mathcal{P}$)=#missing edges inside + #edges outside
  - **NP-hard** to minimize distance to cluster graphs

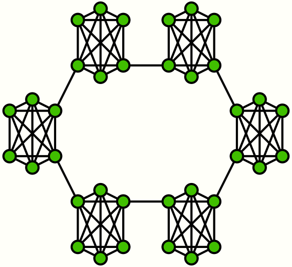
- Constant Potts Model
  - CPM($\mathcal{P}$)=$\sum_{c}[e_c - \gamma\binom{n_c}{2}]$
    where e_c=# edges inside communauty c
    and n_c=# nodes in communauty c
    γ is a chosen constant ≤ 1
  - for $\gamma = 0$?
  - for $\gamma = 1$?
  - for $\gamma = 1/2$?

Is modularity a good quality function ?

- Resolution issue : tends to make too large communities

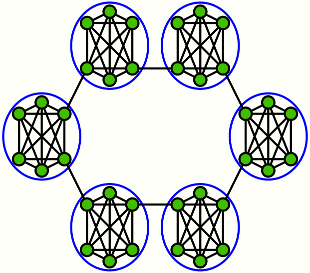
Is modularity a good quality function ?

- Resolution isuue : tends to make too large communities
  Example : ring of p copies of a k-clique ($n = p.k$)



Is modularity a good quality function?

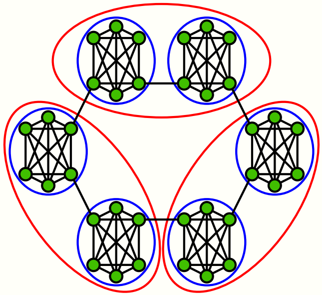
- Resolution isuue : tends to make too large communities
  Example : ring of p copies of a k-clique ($n = p.k$)



$\mathcal{P}_a$ = the cliques

Is modularity a good quality function ?

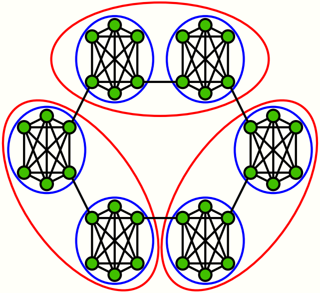
- Resolution isuue : tends to make too large communities
  Example : ring of p copies of a k-clique ($n = p.k$)



$\mathcal{P}_a$ = the cliques
$\mathcal{P}_b$ = the cliques grouped by two

Is modularity a good quality function ?

- Resolution isuue : tends to make too large communities
  Example : ring of p copies of a k-clique ($n = p.k$)

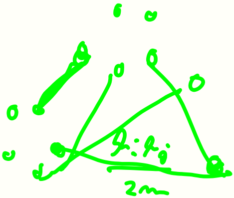


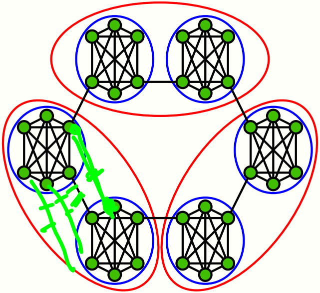
$\mathcal{P}_a$ = the cliques
$\mathcal{P}_b$ = the cliques grouped by two

- Which one is "morally" the best community partition ?

Is modularity a good quality function ?

- Resolution isuue : tends to make too large communities
  Example : ring of p copies of a k-clique ($n = p.k$)





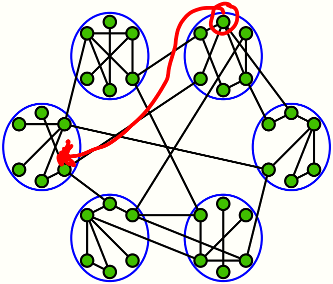
$\mathcal{P}_a$ = the cliques
$\mathcal{P}_b$ = the cliques grouped by two

- Which one is "morally" the best community partition ?
- Which one has higher modularity ?

Louvain algorithm

- **Given a partition**, make a pass through all the vertices :
  - consider each vertex x once in an arbitrary order
  - move x to the community that gives the largest increase in modularity

G (n=30,m=46)

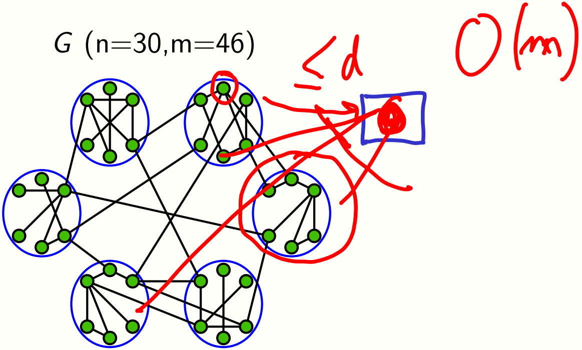


Louvain algorithm

- **Given a partition**, make a pass through all the vertices :
  - consider each vertex x once in an arbitrary order
  - move x to the community that gives the largest increase in modularity

Obs. : non-neighbouring community is never the best

G (n=30,m=46)

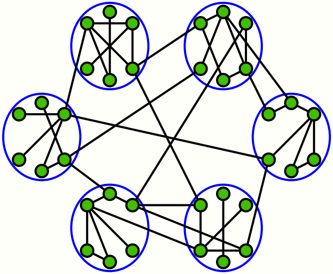


Louvain algorithm

- **Given a partition**, make a pass through all the vertices :
  - consider each vertex x once in an arbitrary order
  - move x to the community that gives the largest increase in modularity (eventually isolated in its own community)

Obs. : non-neighbouring community is never the best

G (n=30,m=46)



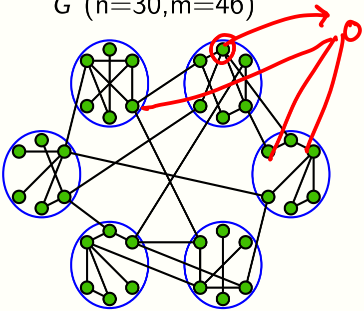
Louvain algorithm

- **Given a partition**, make a pass through all the vertices :
  - consider each vertex x once in an arbitrary order
  - move x to the community that gives the largest increase in modularity (eventually isolated in its own community)

Obs. : non-neighbouring community is never the best

Decompose the move :
- place x alone in its own community
- consider moving x to each neighbourhing community

G (n=30,m=46)



Louvain algorithm

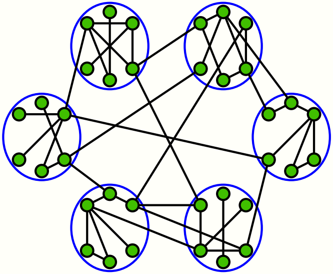
$\rightarrow O(m)$

- **Given a partition**, make a pass through all the vertices :
  - consider each vertex x once in an arbitrary order
  - move x to the community that gives the largest increase in modularity (eventually isolated in its own community)

Obs. : non-neighbouring community is never the best

Decompose the move :
- place x alone in its own community
- consider moving x to each neighbourhing community

G (n=30,m=46)



$$\Delta Q(C,i) = \left[\frac{e_C+k_{i,C}}{2m} - \left(\frac{a_C+k_i}{2m}\right)^2\right] - \left[\frac{e_C}{2m} - \left(\frac{a_C}{2m}\right)^2 - \left(\frac{k_i}{2m}\right)^2\right]$$

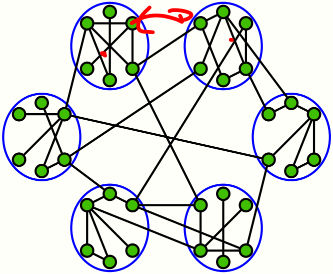
Louvain algorithm

- **Given a partition**, make a pass through all the vertices :
  - consider each vertex x once in an arbitrary order
  - move x to the community that gives the largest increase in modularity (eventually isolated in its own community)

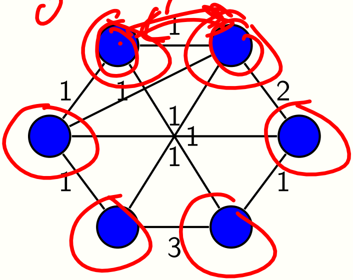
Obs. : non-neighbouring community is never the best

Decompose the move :
- place x alone in its own community
- consider moving x to each neighbourhing community

G (n=30,m=46)



$G/\mathcal{P}$



Louvain algorithm

```
augmented ← true;
while augmented do
    P_0 ← {{x} | x ∈ V(G)} ; P ← P_0 ; Q ← 0;
    while augmented do
        augmented ← faux;
        for i de 1 a n do
            Q_ori ← Q;
            i moves to c_iso = {i} ; Q ← Q − ΔQ_out(i);
            Q_max ← Q ; c_max ← c_iso ;
            for c ∈ P do
                if Q + ΔQ_in(c) > Q_max then
                    Q_max ← Q + ΔQ_in(i, c);
                    c_max ← c;
                end
            end
            If Q_max = Q_ori then c_max ← c_ori else augmented ← true;
            i moves to c_max ; Q ← Q_max ;
        end
    end
    If P ≠ P_0 then augmented ← true ; G ← G/P;
end
return {Expand(P) | P ∈ P};
```

Leiden algorithm

Two improvements over Louvain

- Complexity

Leiden algorithm

Two improvements over Louvain

- Complexity
  - Consider moving only vertices whose neighbours have moved

Leiden algorithm

Two improvements over Louvain

- Complexity
  - Consider moving only vertices whose neighbours have moved
  - Maintain a queue for them

Leiden algorithm

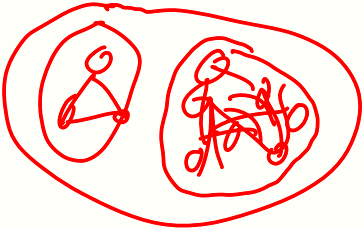
Two improvements over Louvain

- Complexity
  - Consider moving only vertices whose neighbours have moved
  - Maintain a queue for them
  - Same worst case complexity, but better in practice

Leiden algorithm

Two improvements over Louvain

- Complexity
  - Consider moving only vertices whose neighbours have moved
  - Maintain a queue for them
  - Same worst case complexity, but better in practice
- Disconnected (or poorly connected) communities



Leiden algorithm

Two improvements over Louvain

- Complexity
  - Consider moving only vertices whose neighbours have moved
  - Maintain a queue for them
  - Same worst case complexity, but better in practice
- Disconnected (or poorly connected) communities
  - Just before contracting communities, for each community

Leiden algorithm

Two improvements over Louvain

- Complexity
  - Consider moving only vertices whose neighbours have moved
  - Maintain a queue for them
  - Same worst case complexity, but better in practice
- Disconnected (or poorly connected) communities
  - Just before contracting communities, for each community
    - Place vertices alone in their own sub-community

Leiden algorithm

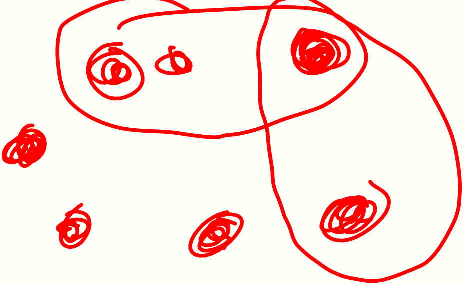
Two improvements over Louvain

- Complexity
  - Consider moving only vertices whose neighbours have moved
  - Maintain a queue for them
  - Same worst case complexity, but better in practice
- Disconnected (or poorly connected) communities
  - Just before contracting communities, for each community
    - Place vertices alone in their own sub-community
    - Merge sub-communities that are strongly connected

Leiden algorithm

Two improvements over Louvain

- Complexity
  - Consider moving only vertices whose neighbours have moved
  - Maintain a queue for them
  - Same worst case complexity, but better in practice
- Disconnected (or poorly connected) communities
  - Just before contracting communities, for each community
    - Place vertices alone in their own sub-community
    - Merge sub-communities that are strongly connected
  - Contract only the obtained sub-communities



Leiden algorithm

Two improvements over Louvain

- Complexity
  - Consider moving only vertices whose neighbours have moved
  - Maintain a queue for them
  - Same worst case complexity, but better in practice
- Disconnected (or poorly connected) communities
  - Just before contracting communities, for each community
    - Place vertices alone in their own sub-community
    - Merge sub-communities that are strongly connected
  - Contract only the obtained sub-communities
  - At the next step start from the partition defined by the whole communities