Lecture 5 - Community detection algorithms
Girvan-Newman, Louvain, Leiden

Autumn 2021 - ENS Lyon

Christophe Crespelle
christophe.crespelle@ens-lyon.fr
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 belongs 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
  
  - \( \sigma_{st} = \# \) 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
  \[ C_B(e) = \sum_{s \neq t} \frac{\sigma_{st}(e)}{\sigma_{st}} \]

  - \( \sigma_{st} = \# \) shortest paths from \( s \) to \( t \)
  - \( \sigma_{st}(e) = \# \) shortest paths from \( s \) to \( t \) containing \( e \)

  - high betweenness \( \iff \) \( e \) 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 \( \iff \) \( e \) is on a high proportion of shortest paths for a high proportion of pairs of nodes

\[ \frac{1}{2} + \frac{1}{2} = 2.5 \]
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 \( \iff \) \( e \) 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 (Girvan-Newman)

- 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 $G$ \( O(m) \)
     - update the betweenness centrality of all links \( O(m^2) \)
  3. output the dendogram of $G$

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

\( n = 10^9 \) \( m = 10^5 \)
The Louvain algorithm

- Idea: optimize a quality function for node partitions

  ▶ modularity: maximize(\(\#\text{edges inside} - \#\text{edges outside}\))
  \(\iff\) maximize(\(\#\text{edges inside}\))
The Louvain algorithm

- Idea: optimize a quality function for node partitions

- Modularity: \[ \text{maximize}(\text{#edges inside} - \text{#edges outside}) \]
  \[ \Leftrightarrow \text{maximize}(\text{#edges inside}) \]

- Problem... the best partition is a single community!!!
The Louvain algorithm

- Idea: optimize a quality function for node partitions

  - modularity: \( \text{maximize}(\#\text{edges inside} - \#\text{edges outside}) \)
  \( \Leftrightarrow \text{maximize}(\#\text{edges inside}) \)

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

- Correction: compare to a randomized version of the network

Original network vs configuration model
Modularity

- Proportion of edges inside communities

\[ Q = \sum_{i} \left( \frac{k_i}{2m} \right) \gamma_{i} \delta(c_i, c_j) \]

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

  $\frac{1}{2m} \sum_{i,j \in V} A_{ij} \delta(c_i, c_j)$ where

original network

configuration model
Modularity

original network

- Proportion of edges inside communities

\[ A \text{ the adjacency matrix of } G \]
\[ k_i \text{ the degree of node } i \]
\[ c_i \text{ the community of node } i \]
\[ \delta \text{ is the Kronecker symbol: } \delta(c_i, c_j) = 1 \text{ iff } c_i = c_j \]

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

- In the configuration model: \[ \frac{1}{2m} \sum_{i,j \in V} \frac{k_i k_j}{2m} \delta(c_i, c_j) \]
Modularity

• Proportion of edges inside communities

\[ A \text{ the adjacency matrix of } G \]
\[ k_i \text{ the degree of node } i \]
\[ c_i \text{ the community of node } i \]
\[ \delta \text{ is the Kronecker symbol: } \delta(c_i, c_j) = 1 \text{ iff } c_i = c_j \]

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

- In the configuration model:
  \[ \frac{1}{2m} \sum_{i,j \in V} k_i k_j \delta(c_i, c_j) \]

• modularity: \[ Q(P) = \frac{1}{2m} \sum_{i,j \in V} [A_{ij} - \frac{k_i k_j}{2m}] \delta(c_i, c_j) \]
  \[ = \frac{1}{2m} \sum_{c \in P} [e_c - \frac{\delta_c^2}{2m}] \]
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 \( i \)
\( \delta \) is the Kronecker symbol: \( \delta(c_i, c_j) = 1 \) iff \( c_i = c_j \)

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

- 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 dendrogram produced by Girvan-Newman
Other quality functions

• Distance to cluster graphs

$$\text{dist-cluster}(P) = \#\text{missing edges inside} + \#\text{edges outside}$$
Other quality functions

- Distance to cluster graphs

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

- \textbf{NP-hard} to minimize distance to cluster graphs
Other quality functions

- **Distance to cluster graphs**
  
  \[ \text{dist-cluster}(\mathcal{P}) = \#\text{missing edges inside} + \#\text{edges outside} \]

  **NP-hard** to minimize distance to cluster graphs

- **Constant Potts Model**
  
  \[ \text{CPM}(\mathcal{P}) = \sum_c [e_c - \gamma \left( \binom{n_c}{2} \right)] \]

  where \( e_c = \# \text{ edges inside community } c \)
  
  and \( n_c = \# \text{ nodes in community } c \)

  \( \gamma \) is a chosen constant \( \leq 1 \)
Other quality functions

- Distance to cluster graphs
  \[ \text{dist-cluster}(\mathcal{P}) = \# \text{missing edges inside} + \# \text{edges outside} \]
  \( \text{NP-hard} \) to minimize distance to cluster graphs

- Constant Potts Model
  \[ \text{CPM}(\mathcal{P}) = \sum_c [e_c - \gamma \binom{n_c}{2}] \]
  where \( e_c = \# \text{ edges inside community } c \)
  and \( n_c = \# \text{ nodes in community } c \)
  \( \gamma \) is a chosen constant \( \leq 1 \)
  for \( \gamma = 0 \)?
Other quality functions

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

- Constant Potts Model
  - $\text{CPM}(\mathcal{P}) = \sum_c [e_c - \gamma \binom{n_c}{2}]$
    - where $e_c = \# \text{edges inside community } c$
    - and $n_c = \# \text{nodes in community } c$
    - $\gamma$ is a chosen constant $\leq 1$
    - for $\gamma = 0$ → all the nodes connected
    - for $\gamma = 1$ → all the nodes isolated.
Other quality functions

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

- **Constant Potts Model**
  
  \[ \text{CPM}(\mathcal{P}) = \sum_c [e_c - \gamma \left( \frac{n_c}{2} \right)] \]
  
  where \( e_c = \# \text{edges inside community } c \)
  
  and \( n_c = \# \text{nodes in community } c \)
  
  \( \gamma \) is a chosen constant \( \leq 1 \)
  
  - 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 issue: tends to make too large communities
  Example: ring of $p$ copies of a $k$-clique ($n = p \cdot k$)
Is modularity a good quality function?

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

\[ \mathcal{P}_a = \text{the cliques} \]
Is modularity a good quality function?

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

$$\mathcal{P}_a = \text{the cliques}$$

$$\mathcal{P}_b = \text{the cliques grouped by two}$$
Is modularity a good quality function?

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

$\mathcal{P}_a = \text{the cliques}$

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

Which one is "morally" the best community partition?
Is modularity a good quality function?

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

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

▸ 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
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 neighbouring community

$G \ (n=30, m=46)$
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 neighbouring community

$G \ (n=30, m=46)$

$$\Delta Q(C, i) = \left( \frac{eC + k_i}{2m} - \left( \frac{aC + k_i}{2m} \right)^2 \right) - \left( \frac{eC}{2m} - \left( \frac{aC}{2m} \right)^2 - \left( \frac{k_i}{2m} \right)^2 \right)$$
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 neighbouring community
Louvain algorithm

```
1 augmented ← true;
2 while augmented do
3     P₀ ← {{x} | x ∈ V(G)}; P ← P₀; Q ← 0;
4     while augmented do
5         augmented ← faux;
6         for i de 1 a n do
7             Qori ← Q;
8             i moves to c_iso = {i}; Q ← Q − ΔQ_out(i);
9             Q_max ← Q; c_max ← c_iso;
10            for c ∈ P do
11                if Q + ΔQ_in(c) > Q_max then
12                    Q_max ← Q + ΔQ_in(i, c);
13                    c_max ← c;
14                end
15            end
16            If Q_max = Qori then c_max ← c_ori else augmented ← true;
17            i moves to c_max; Q ← Q_max;
18        end
19    end
20    if P ≠ P₀ then augmented ← true; G ← G/P;
21 end
22 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