

# **Massive Graph Management & Analytics**

## **COMMUNITY DETECTION APPROACHES**

Nacéra Seghouani

Computer Science Department, CentraleSupélec  
Laboratoire Interdisciplinaire des Sciences du Numérique, LISN  
[nacera.seghouani@centralesupelec.fr](mailto:nacera.seghouani@centralesupelec.fr)

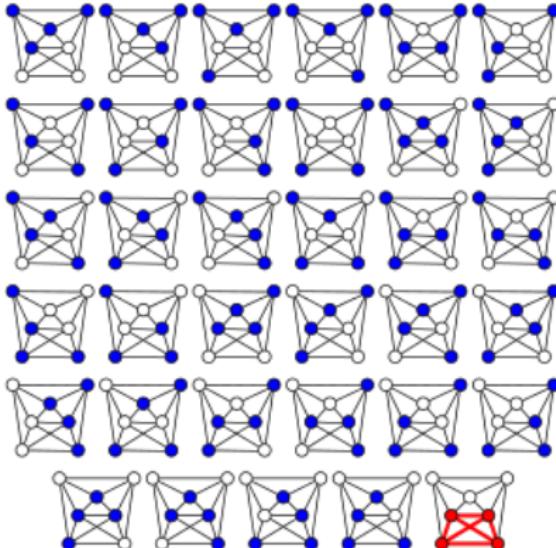
2023-2024

# Outline

- ☞ *k*-Clique and Maximal Cliques
  - Bron-Kerbosch Algorithm
- ☞ *k*-Core Subgraphs Decomposition
  - *k*-Core Subgraphs Decomposition
- ☞ Community Detection Problem
  - Clique-based approach
  - Modularity maximisation approach
  - Random walk-based approach
  - Edge betweenness separation-based approach
- ☞ Research paper presentation

## Typical large network structure: $k$ -Clique

- A clique is a complete subgraph.  $k$ -clique is a complete subgraph (subset of vertices, all adjacent to each other) with  $k$  nodes
- Brute force algorithm finds all 4-cliques in this 7-node graph  :  $C_7^4$  combinations for completeness.



# Typical large network structure: $k$ -Clique

- ☞ Most versions of clique problems are hard.
- ☞ Common formulation problem: finding maximal cliques, i.e. cliques with the largest number of nodes.
  - ✓ Many algorithms: Bron–Kerbosch algorithm to find all maximal cliques (different variants,  $O(3^{\frac{n}{3}})$  in the worst case), Tarjan & Trojanowski algorithm and more recently Janez Konc algorithm.
  - ✓ Many algorithms with better theoretical complexity to fin all maximal cliques, Bron–Kerbosch (and improvements) are more efficient.
- ☞ Relaxation of a clique:  $k$ -core (seen later)

# Bron-Kerbosch Algorithm: All Maximal Cliques

- Basic form of Bron–Kerbosch algorithm:

\*\*\* $R$ : nodes of a potential clique;  $P$ : nodes to be added to a potential clique;  $X$ : nodes already processed or already belong to a maximal clique\*\*\*

$P = V, X = \emptyset, R = \emptyset$

*BronKerbosch( $R, P, X$ )*

**if**  $P = \emptyset$  and  $X = \emptyset$  **then**

$R$  is a maximal clique

**end if**

**for**  $v \in P$  **do**

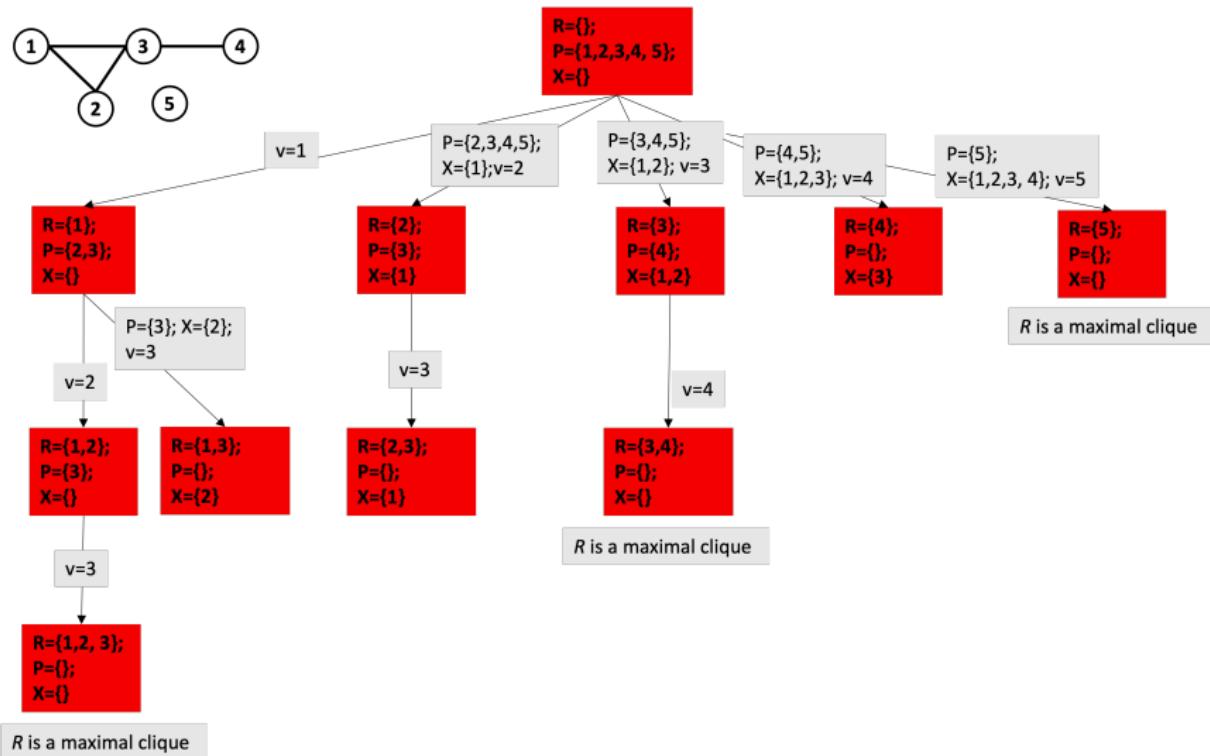
*BronKerbosch( $R \cup \{v\}, P \cap \mathcal{N}_v, X \cap \mathcal{N}_v$ )*

$P \leftarrow P \setminus \{v\}$

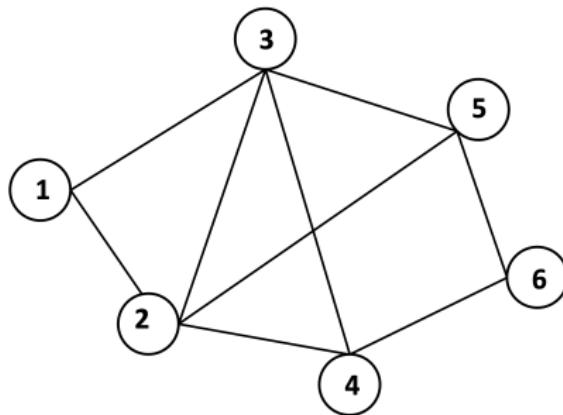
$X \leftarrow X \cup \{v\}$

**end for**

# Bron-Kerbosch Algorithm: All Maximal Cliques - Example

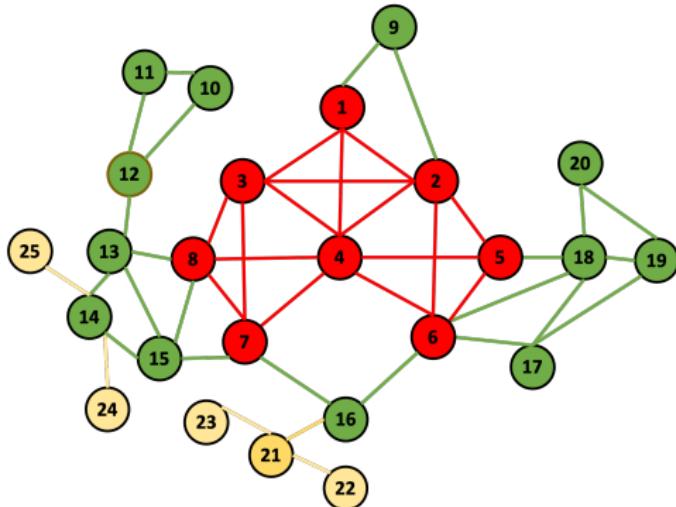


# Bron-Kerbosch Algorithm: All Maximal Cliques - Exercise



# Typical large network structure: $k$ -Core

- Core-periphery structure of a network: internal core where a lot of nodes are connected and peripheral nodes called whiskers (Leskovec et al, 2009).



- $k$ -core of a graph  $G$  is a maximal connected subgraph of  $G$  where vertices have at least  $k$ -degree. Also called  $k$ -degenerate.

# Typical large network structure: $k$ -Core

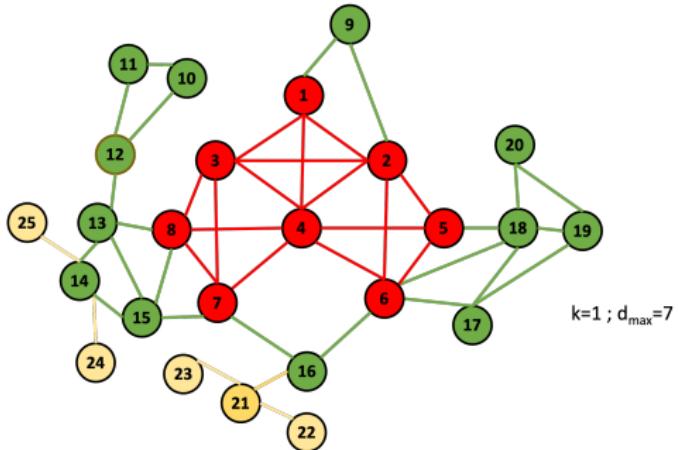
- $k$ -core decomposition: Given a graph  $\mathcal{G}(V, E)$ , delete recursively all vertices, and edges connecting them, of degree less than  $k$  to extract the  $k$ -core graph such that:
  - ✓ Each  $v_i$  in a  $k$ -core graph has  $d_i \geq k$
  - ✓  $(k+1)$ -core is a subgraph of a  $k$ -core graph

- Matula & Beck, 1983;

```
1: procedure  $\text{kCORE}(\mathcal{G}(V, E))$ 
2:    $L[ ]; d[ ]; D[ \{ \}]$ ;  $d_{\max} = \max_{v_i \in V} d_{v_i}$ 
3:   **  $L$  list of ranked  $k$  by node,  $d$  list of degrees by node,  $D$  list of set of nodes by degree
4:   for  $v_i \in V$  do
5:      $d[i] \leftarrow d_{v_i}$ ;  $D[d[i]].insert(v_i)$ 
6:   end for
7:   for  $k \leftarrow 0$  to  $d_{\max}$  do
8:     while  $D[k].notempty()$  do
9:        $i \leftarrow D[k].remove(); L[i] \leftarrow k$ 
10:      for  $v_j \in \mathcal{N}_{v_i}$  do
11:        if  $d[j] > k$  then
12:           $D[d[j]].remove(j); D[d[j] - 1].insert(j); d[j] --$ 
13:        end if
14:      end for
15:    end while
16:  end for
17: end procedure
```

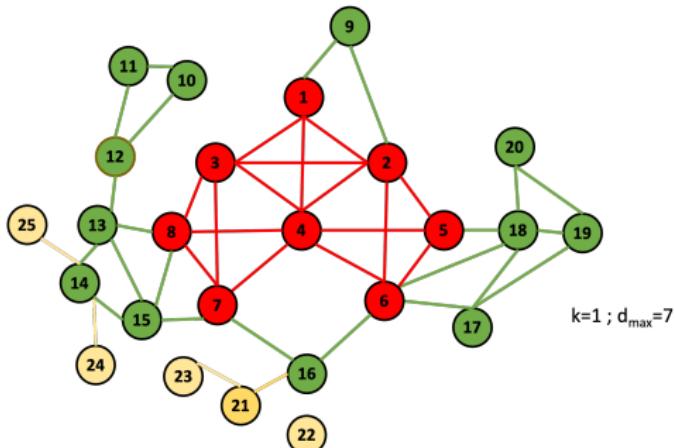


# Typical large network structure: $k$ -Core



$d[v_i]$	7	6	5	4	3	2	1
$D[d[v_i]]$	{4}	{2,6}	{3,7,8,18}	{1,5,13,14,15}	{12,16,17,19,21}	{9,10,11,20}	{22,23,24,25}

# Typical large network structure: $k$ -Core

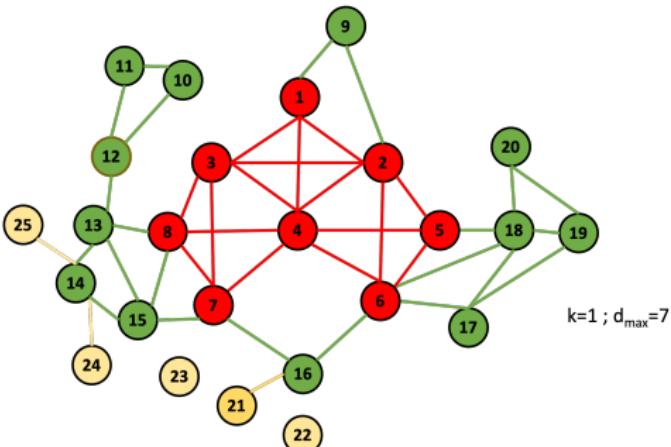


i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
$d[v_i]$	4	6	5	7	4	6	5	5	2	2	2	3	4	4	4	3	3	5	3	2	2	1	1	1	1

$d[v_i]$	7	6	5	4	3	2
$D[d[v_i]]$	{4}	{2,6}	{3,7,8,18}	{1, 5, 13, 14, 15}	{12, 16, 17, 19, 21}	{9, 10, 11, 20, 21}

$v_i$	22
$L[v_i]$	1

# Typical large network structure: $k$ -Core

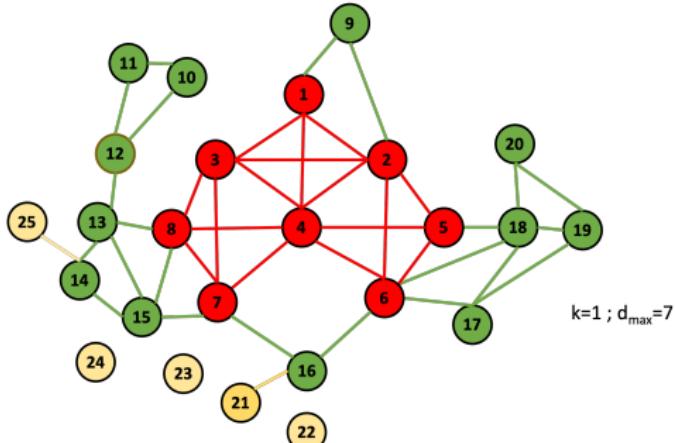


i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
$d[v_i]$	4	6	5	7	4	6	5	5	2	2	2	3	4	4	4	3	3	5	3	2	1	1	1	1	1

$d[v_i]$	7	6	5	4	3	2	1
$D[d[v_i]]$	{4}	{2,6}	{3,7,8,18}	{1, 5, 13, 14, 15}	{12, 16, 17, 19}	{9, 10, 11, 20, 24}	{28, 24, 25, 21}

$v_i$	22	23
$L[v_i]$	1	1

# Typical large network structure: $k$ -Core

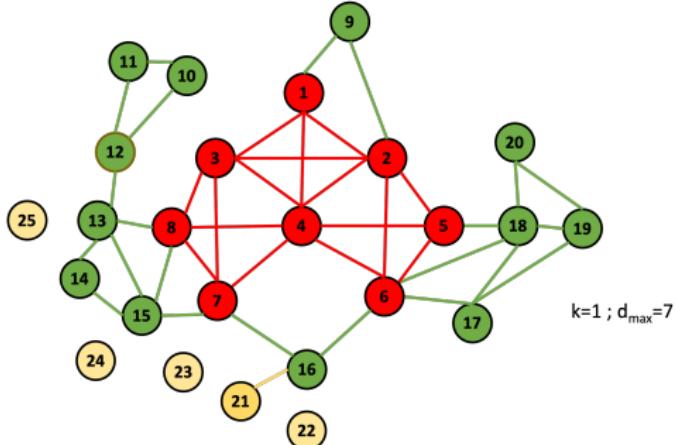


i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
$d[v_i]$	4	6	5	7	4	6	5	5	2	2	2	3	4	3	4	3	3	5	3	2	1	1	1	1	1

$d[v_i]$	7	6	5	4	3	2	1
$D[d[v_i]]$	{4}	{2,6}	{3,7,8,18}	{1, 5,13,14,15}	{12, 16, 17, 19, 14}	{9, 10, 11, 20}	{24, 25, 21}

$v_i$	22	23	24
$L[v_i]$	1	1	1

# Typical large network structure: $k$ -Core

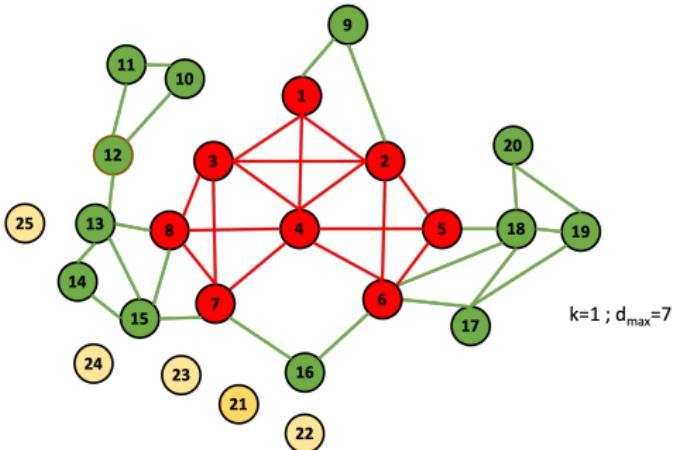


i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
$d[v_i]$	4	6	5	7	4	6	5	5	2	2	2	3	4	2	4	3	3	5	3	2	1	1	1	1	1

$d[v_i]$	7	6	5	4	3	2	
$D[d[v_i]]$	{4}	{2,6}	{3,7,8,18}	{1,5,13,15}	{12, 16, 17, 19, 21}	{9, 10, 11, 20, 21}	{25, 21}

$v_i$	22	23	24	25
$L[v_i]$	1	1	1	1

# Typical large network structure: $k$ -Core

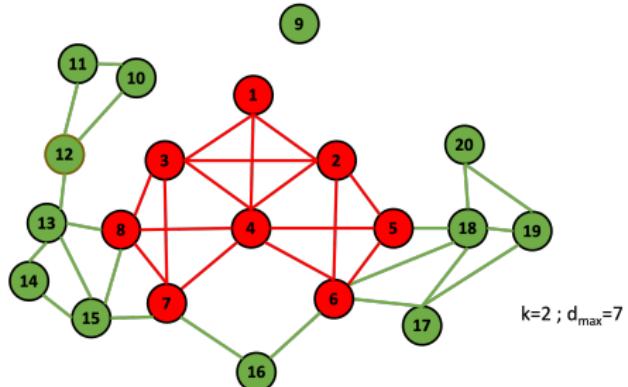


i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
$d[v_i]$	4	6	5	7	4	6	5	5	2	2	2	3	4	2	4	2	2	5	3	2	1	1	1	1	1

$d[v_i]$	7	6	5	4	3	2
$D[d[v_i]]$	{4}	{2,6}	{3,7,8,18}	{1,5,13,15}	{12,16,17,19}	{9,10,11,20,14,16}

$v_i$	22	23	24	25	21
$l[v_i]$	1	1	1	1	1

# Typical large network structure: $k$ -Core



i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
$d[v_i]$	3	5	5	7	4	6	5	5	2	2	2	3	4	2	4	3	2	5	3	2	1	1	1	1	1

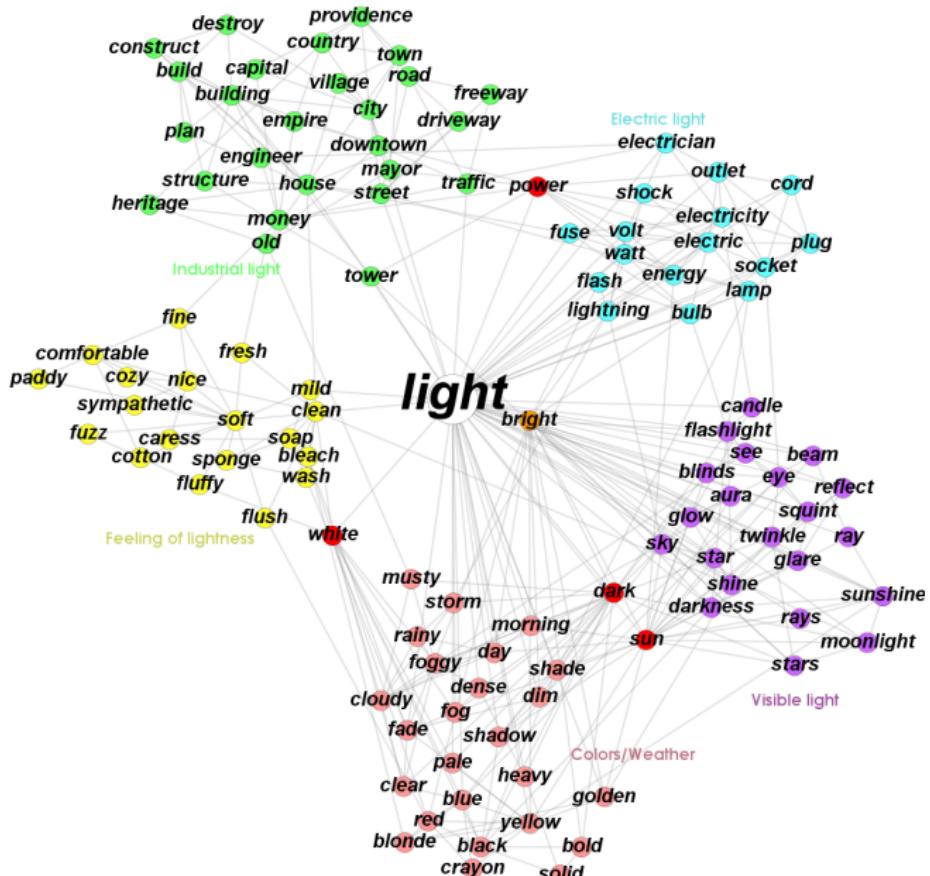
$d[v_i]$	7	6	5	4	3	2
$D[d[v_i]]$	{4}	{5, 6}	{3, 7, 8, 18, 2}	{1, 5, 13, 15}	{12, 17, 19, 1}	{9, 10, 11, 20, 14, 16}

$v_i$	22	23	24	25	21	9
$L[v_i]$	1	1	1	1	1	2

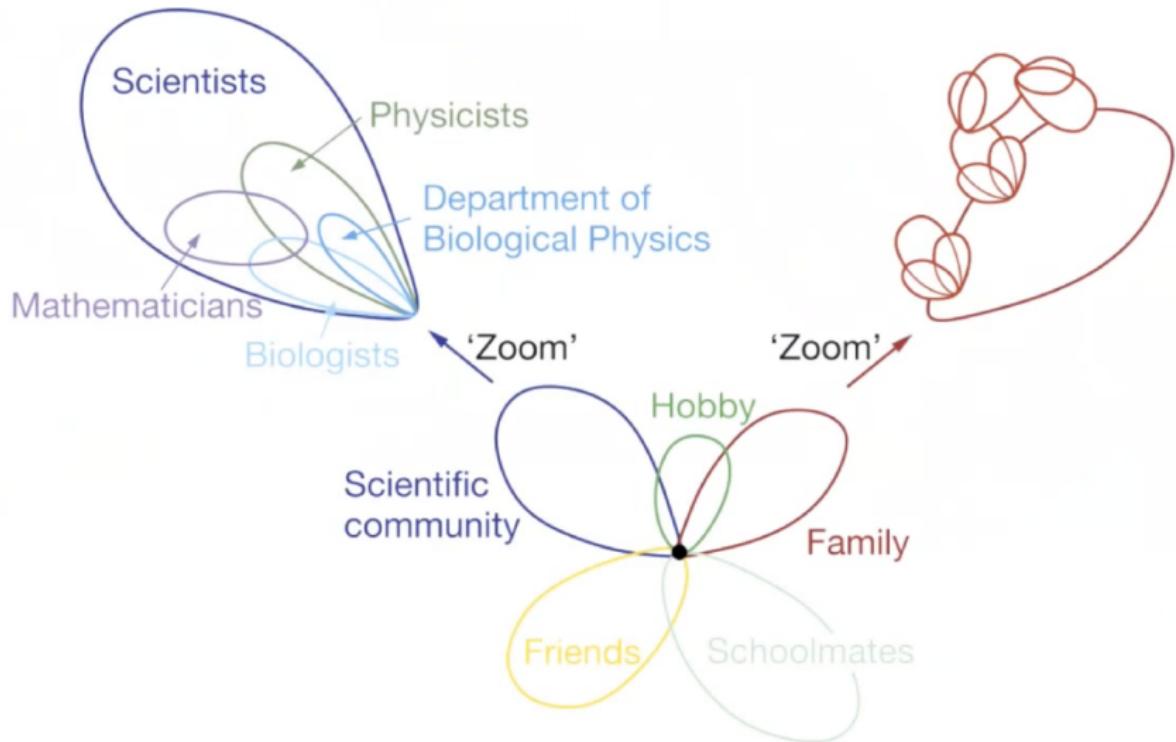
# Community Detection Problem

- 👉 Community in network science refers to the occurrence of clusters (groups) of nodes in a network that are more densely connected internally than with the other nodes
- 👉 Two cases:
  - ✓ Non-overlapping community: divides into clusters of nodes with dense connections internally and sparser connections between clusters. Each node assigned to one community.
  - ✓ Overlapping community: assumes that pairs of nodes are more likely to be connected if they are both members of the same communities, and less likely to be connected if they do not share any community.

# Non-overlapping Example



# Overlapping Example



# Community Detection Problem

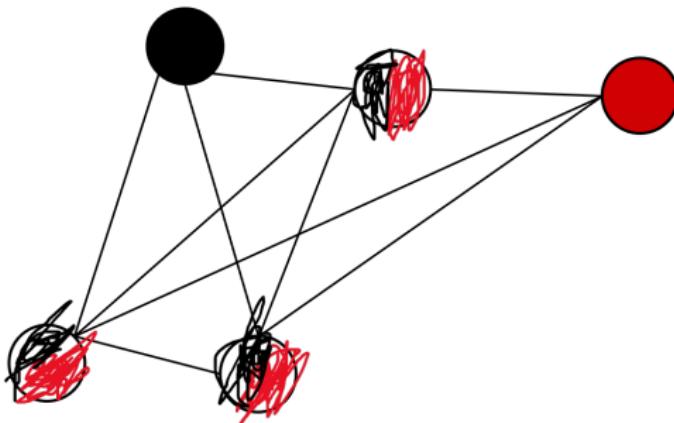
- ✓ Mutuality of ties: every node in the group has ties to one another
- ✓ Compactness: closeness or reachability of group members
- ✓ Density of edges: high number of edges within a group
- ✓ Separation higher frequency of ties among group members compared to non members
- ✓ Usual metrics: Graph density, internal and external density, graph cut, modularity score

# Community Detection Problem

- ☞ Combinatorial optimisation problem, according to some metrics. Exact solution NP-hard
- ☞ Many approaches by greedy or approximate heuristics for community detection in the literature: agglomerative/hierarchical (ex. Dendrogram) or partitioning (Newman-Girvan 2004 ↗)
  - ✓ Clique-based approach
  - ✓ Modularity maximisation approach
  - ✓ Random walk-based approach
  - ✓ Edge betweenness separation-based approach

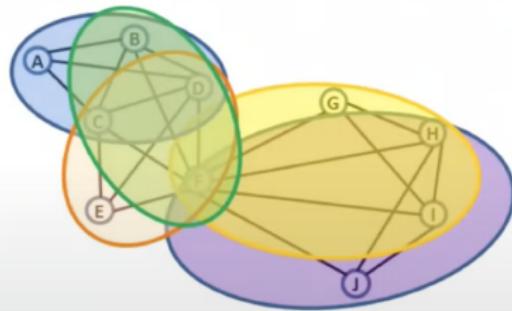
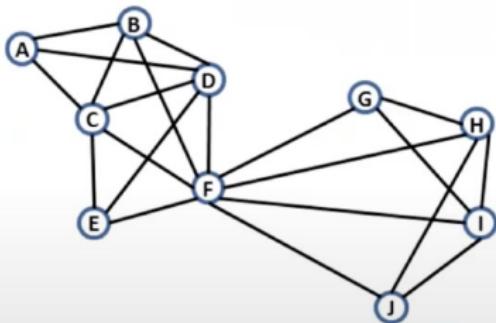
## ***k*-Clique Community**

- ☞ A  $k$ -clique community is the union of all  $k$ -cliques that can be reached from one to the other through a sequence of adjacent  $k$ -cliques.
- ☞ Two  $k$ -cliques are adjacent if they share  $k - 1$  nodes.



# **k-Clique Community**

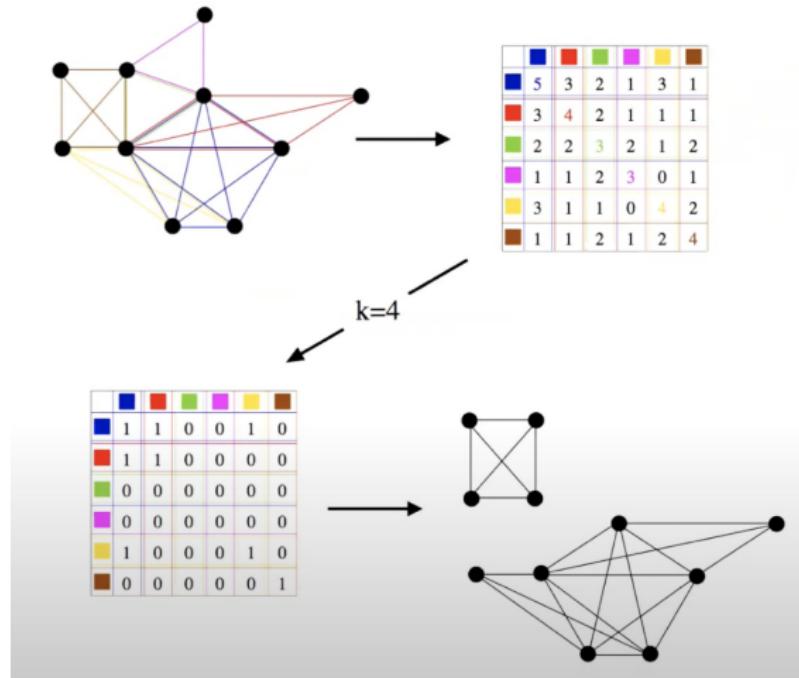
- Blue community  $\{A, B, C, D\}$  is adjacent to green community  $\{B, C, D, F\}$
- Green community  $\{C, D, F, B\}$  is adjacent to orange community  $\{C, D, F, E\}$
- Purple community is adjacent  $\{F, H, I, J\}$  to yellow community  $\{F, H, I, G\}$



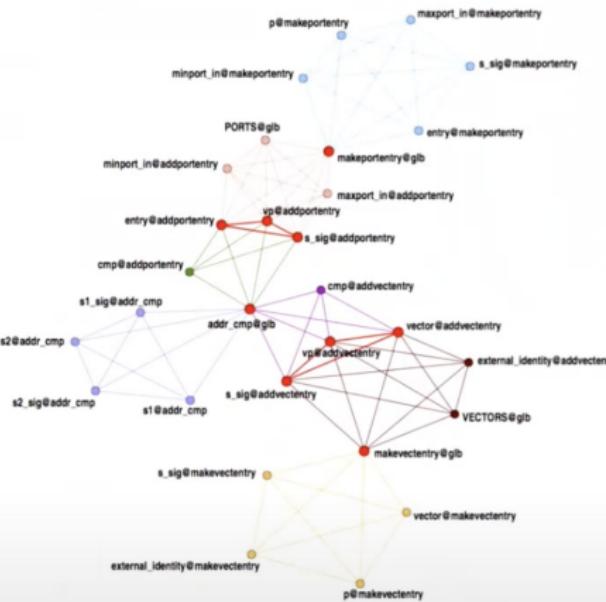
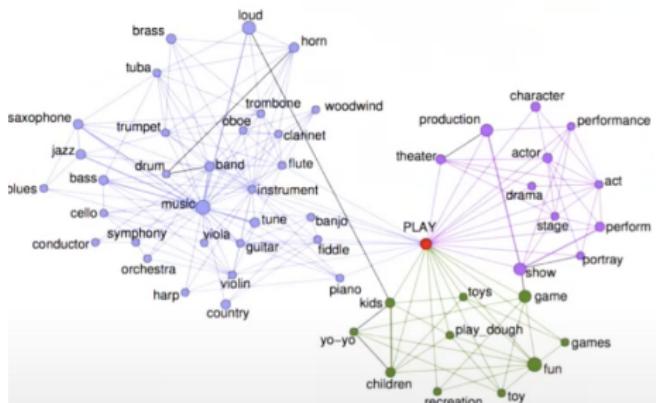
Adjacent 4-cliques

## Clique Percolation Method: Algorithm

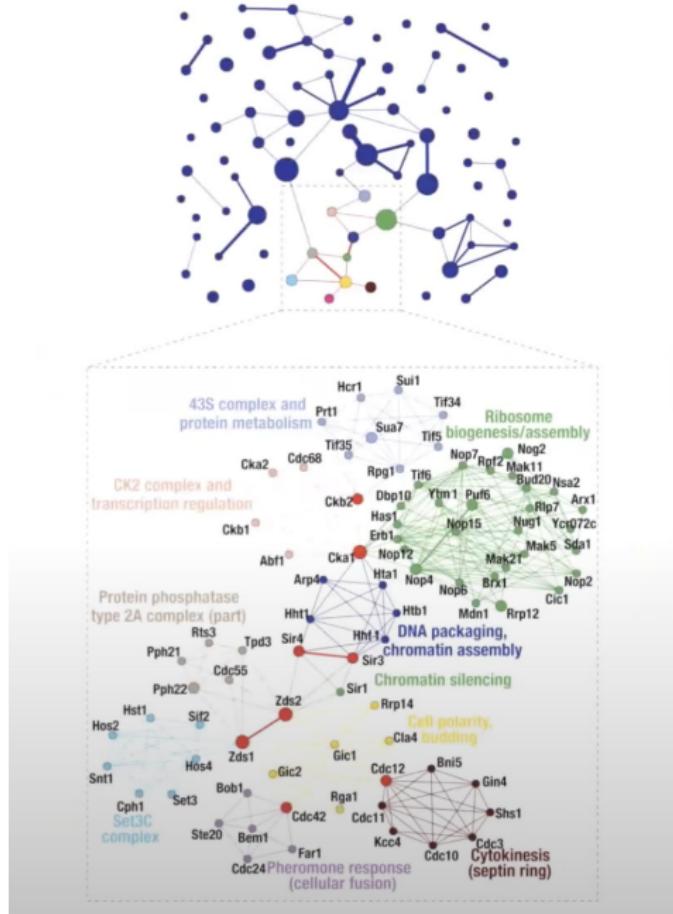
- ✓ Palla et al. 2005 ↗
  - ✓ Find all maximal cliques, sorting them based on degrees.
  - ✓ Create clique overlap matrix at a given  $k - 1$  how many nodes are overlapping.
  - ✓ Communities are connected components: blue+red+yellow and brown.



## Clique Percolation Method: Example $k = 4, k = 5$

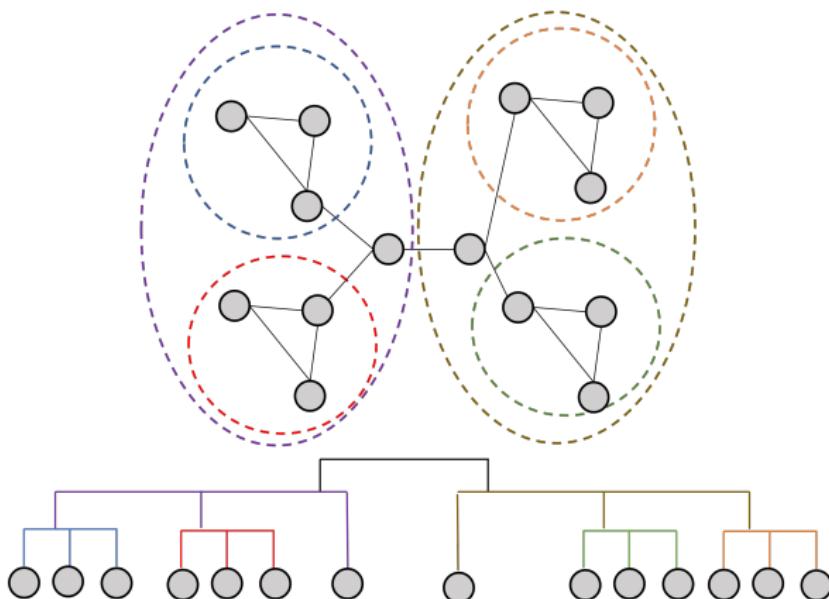


## Clique Percolation Method:



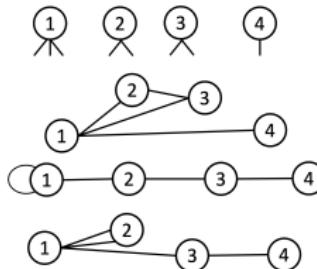
# Community Detection: Louvain Algorithm

- ✓ Louvain: The most popular community detection based on modularity measure. Widely used for large networks: efficient (fast convergence), high dense communities (Blondel et al, 2008) ↗.
- ✓ Greedy algorithm with  $O(n \log n)$  in time
- ✓ Supports weighted graphs
- ✓ Provides hierarchical communities



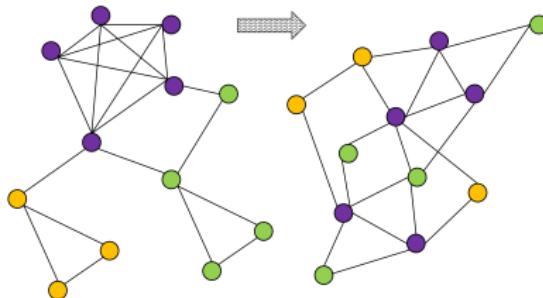
# Community Detection: Modularity

- ✓ In network science, the configuration model is a method for generating random networks from a given degree sequence.
  - ☞ Widely used as a reference model for social networks, it allows the modeler to incorporate arbitrary degree distributions.
  - ☞ Assign a degree  $k_v$  to each node  $v$ . Each degree is a half-link or stub. The sum of stubs must be even to be able to build a graph  $\sum_{v \in V} d_v = 2m$ .
  - ☞ Choose two stubs uniformly at random and connect them by a link. Choose another pair from the remaining  $2m - 2$  stubs and connect them. Continue until you run out of stubs.
  - ☞ The resulting network keeps the same degrees but randomly pair up nodes.
- ✓ A realization might include cycles, self-loops or multi-links. The uniform distribution of the matching must be kept. This is why it does not exclude self-loops or multi-links. Their expected number goes to zero for large networks.
- ✓ The probability of a stub of  $v$  being connected to one of  $w$  stubs is  $\frac{d_w}{2m-1}$ . Since node  $v$  has  $d_v$  stubs, the probability of  $v$  being connected to  $w$  is  $\frac{d_v d_w}{2m-1} \approx \frac{d_v d_w}{2m}$  for large  $m$



# Community Detection: Modularity

- ✓ Modularity measures the relative density of links inside communities with respect to links outside communities. Different methods to compute the modularity. In the most common version, the randomization of the edges is done so as to preserve the degree of each vertex.
- ✓ The basic idea is to compare the number of links within communities with the number expected on the basis of chance. The generated network has less links between nodes of the same community and more between nodes of different communities.



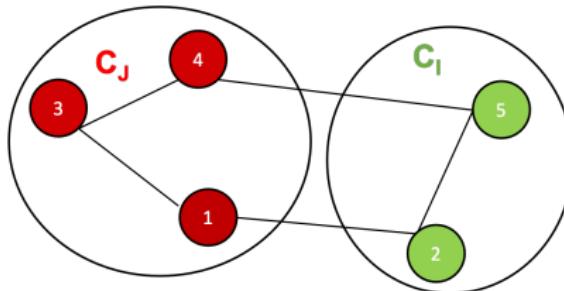
$$Q = \frac{1}{2m} \sum_i \sum_j \left[ A_{ij} - \frac{k_i k_j}{2m} \right] \delta(c_i, c_j) \text{ or } Q = \left[ \frac{\sum_i \sum_j A_{ij}}{2m} - \frac{\sum_i k_i \sum_j k_j}{(2m)^2} \right] \delta(c_i, c_j)$$

where  $k_i$  and  $k_j$  are resp. the sum of weights of  $v_i$  and  $v_j$  links (degree if unweighted graph), and  $c_i$  and  $c_j$  are resp. the communities of  $v_i$  and  $v_j$ . Kronecker function  $\delta$ : 1 if  $c_i = c_j$ ; 0 otherwise.

$-1 \leq Q \leq 1$ .  $Q > 0$ : when links within communities > links within communities in a randomly rewired network.

# Community Detection: Modularity

Compare fraction of edges within a community to expected edges were distributed at random.  $\frac{k_i k_j}{2m}$  is the random distribution of a link on the original graph. For the entire graph we sum the modularities over all communities.



$\delta$ : 1 if  $c_i = c_j$ ; 0 otherwise

$$A_{11} - \frac{k_1 k_1}{2 * 5} = \left(0 - \frac{2 * 2}{2 * 5}\right) * 1 = -\frac{2}{5}; A_{33} - \frac{k_3 k_3}{2 * 5} = \left(0 - \frac{2 * 2}{2 * 5}\right) * 1 = -\frac{2}{5}; A_{44} - \frac{k_4 k_4}{2 * 5} = \left(0 - \frac{2 * 2}{2 * 5}\right) * 1 = -\frac{2}{5}$$

$$A_{13} - \frac{k_1 k_3}{2 * 5} = \left(1 - \frac{2 * 2}{2 * 5}\right) * 1 = \frac{3}{5}; A_{14} - \frac{k_1 k_4}{2 * 5} = \left(0 - \frac{2 * 2}{2 * 5}\right) * 1 = -\frac{2}{5}; A_{34} - \frac{k_3 k_4}{2 * 5} = \left(1 - \frac{2 * 2}{2 * 5}\right) * 1 = \frac{3}{5}$$

$$A_{55} - \frac{k_5 k_5}{2 * 5} = \left(0 - \frac{2 * 2}{2 * 5}\right) * 1 = -\frac{2}{5}; A_{22} - \frac{k_2 k_2}{2 * 5} = \left(0 - \frac{2 * 2}{2 * 5}\right) * 1 = -\frac{2}{5}; A_{25} - \frac{k_2 k_5}{2 * 5} = \left(1 - \frac{2 * 2}{2 * 5}\right) * 1 = \frac{3}{5};$$

$$Q = \frac{1}{2 * 5} \left[ -\frac{2}{5} - \frac{2}{5} - \frac{2}{5} + 2 * \frac{3}{5} - 2 * \frac{2}{5} + 2 * \frac{3}{5} - \frac{2}{5} - \frac{2}{5} + 2 * \frac{3}{5} \right] = \frac{2}{25}$$

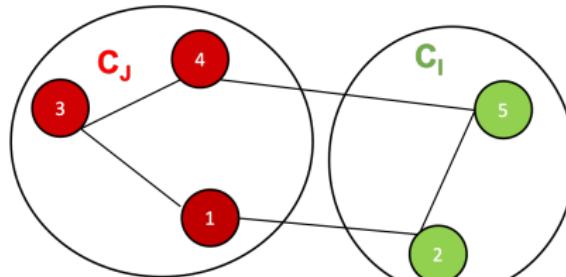
# Community Detection: Modularity

The modularity of a community  $c$ :

$$Q_c = \frac{\sum_{i \in c} \sum_{j \in c} A_{ij}}{2m} - \frac{\sum_{i \in c} k_i \sum_{j \in c} k_j}{(2m)^2}$$

$$Q_c = \frac{\sum_{in}}{2m} - \left( \frac{\sum_{in+out}}{2m} \right)^2$$

where  $\sum_{in}$  is the sum of edge weights between nodes within the community  $c$  (each edge is considered twice); and  $\sum_{in+out}$  is the sum of all edge weights for nodes within the community; including incident ones from the other communities. If the number of communities = 1,  $Q = 0$ .

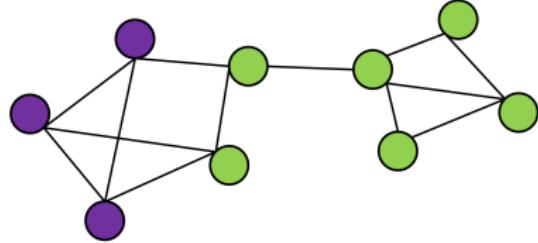
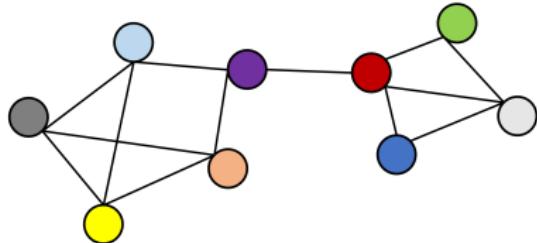
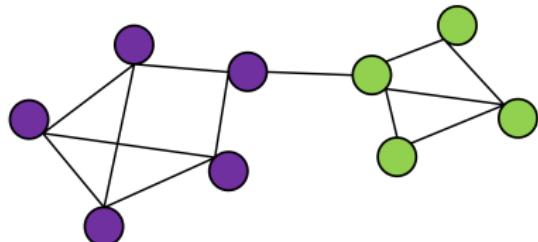
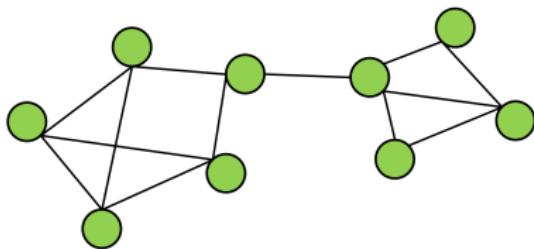


$$Q_{C_J} = \frac{2*2}{2*5} - \left( \frac{2*2+2}{2*5} \right)^2; Q_{C_I} = \frac{2*1}{2*5} - \left( \frac{2*1+2}{2*5} \right)^2$$

$$Q = \frac{2}{25}$$

# Community Detection: Louvain Algorithm

Compare fraction of edges within a community to expected edges were distributed at random.  $\frac{k_i k_j}{2m}$  is the random distribution of a link on the original graph. For the entire graph we sum the modularities over all communities.



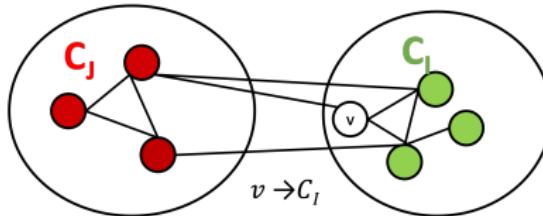
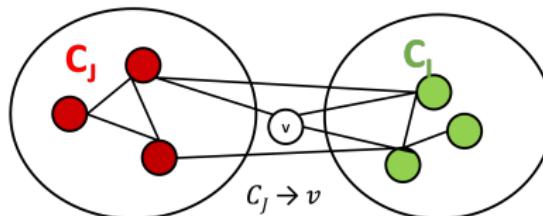
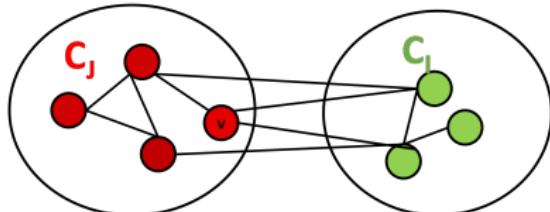
# Community Detection: Louvain method

- ✓ Louvain method: heuristic method for greedy modularity maximisation in 2 phases:
- ✓ Optimal value of modularity is impractical because it needs to go through all possible iterations of the nodes into groups
  - Greedy heuristic: first small communities are found by optimizing modularity locally on all nodes, then each small community is grouped into one node and the first step is repeated.
- ✓ Phase 1: Modularity is optimized by allowing only local changes to node communities memberships
  - Initially each node is assigned its own community.
- ✓ Phase 2: The identified communities are aggregated into super-nodes to build a new network
- ✓ Repeat Phase 1 until 1 super node

# Community Detection: Louvain Algorithm

## Phase 1

- ✓ What is  $\Delta Q$  if we move a node  $v$  from  $C_J$  to  $C_I$ :  $\Delta Q_{C_J \rightarrow v \rightarrow C_I} = \Delta Q_{C_J / \{v\}} + \Delta Q_{C_I \cup \{v\}}$



# Community Detection: Louvain Algorithm

$$\Delta Q_{C_J \rightarrow v \rightarrow C_I} = \Delta Q_{C_J / \{v\}} + \Delta Q_{C_I \cup \{v\}}$$

- ✓ How we can derive  $\Delta Q_{C_I \cup \{v\}}$ ? the gain in modularity doesn't depend on the original community of  $v$ . It can easily be computed by moving an isolated node  $v$  into  $C_I$ :

$$\Delta Q_{C_I \cup \{v\}} = Q_{C_I \cup \{v\}} - [Q_{C_I} + Q_{\{v\}}]$$

$$\Delta Q_{C_I \cup \{v\}} = \left[ \frac{\sum_{in} + 2k_{v,in}}{2m} - \left( \frac{\sum_{in+out} + k_v}{2m} \right)^2 \right] - \left[ \frac{\sum_{in}}{2m} - \left( \frac{\sum_{in+out}}{2m} \right)^2 - \left( \frac{k_v}{2m} \right)^2 \right]$$

$$\Delta Q_{C_J \setminus \{v\}} = [Q_{C_J \setminus \{v\}} + Q_{\{v\}}] - Q_{C_J}$$

$$\Delta Q_{C_J \setminus \{v\}} = \left[ \frac{\sum_{in} - 2k_{v,in}}{2m} - \left( \frac{\sum_{in+out} - k_v}{2m} \right)^2 - \left( \frac{k_v}{2m} \right)^2 \right] - \left[ \frac{\sum_{in}}{2m} - \left( \frac{\sum_{in+out}}{2m} \right)^2 \right]$$

- ✓ The gain computation depends only on  $k_{v,in}$  and  $C$

# Community Detection: Louvain Algorithm

- For each node  $v$ : (i) compute modularity gain from removing  $v$  from its community and placing it in the community of its neighbor; (ii) place  $v$  in the community maximizing  $\Delta Q$ . One iteration is achieved for all the nodes in a sequential manner.

```
1: procedure LOUVAIN-PHASE1-ONEITERATION( $\mathcal{G}(V, E, K), C$ )
2:   for  $v \in V$  do
3:      $cv \leftarrow C.get(v)$ ;  $Cn \leftarrow C.get(\mathcal{N}_v) \setminus cv$ ;  $gmax \leftarrow -\infty$ ;  $cvmax \leftarrow cv$ 
4:     for  $c \in Cn$  do
5:        $gain \leftarrow \Delta Q_{cv \rightarrow v \rightarrow c}$ 
6:       if  $gain > gmax$  then
7:          $gmax \leftarrow gain$ ;  $cvmax \leftarrow c$ 
8:       end if
9:     end for
10:    if  $gmax > 0$  then
11:       $cv \leftarrow cv \setminus \{v\}$ ;  $cvmax \leftarrow cvmax \cup \{v\}$ 
12:    end if
13:  end for
14: end procedure
```

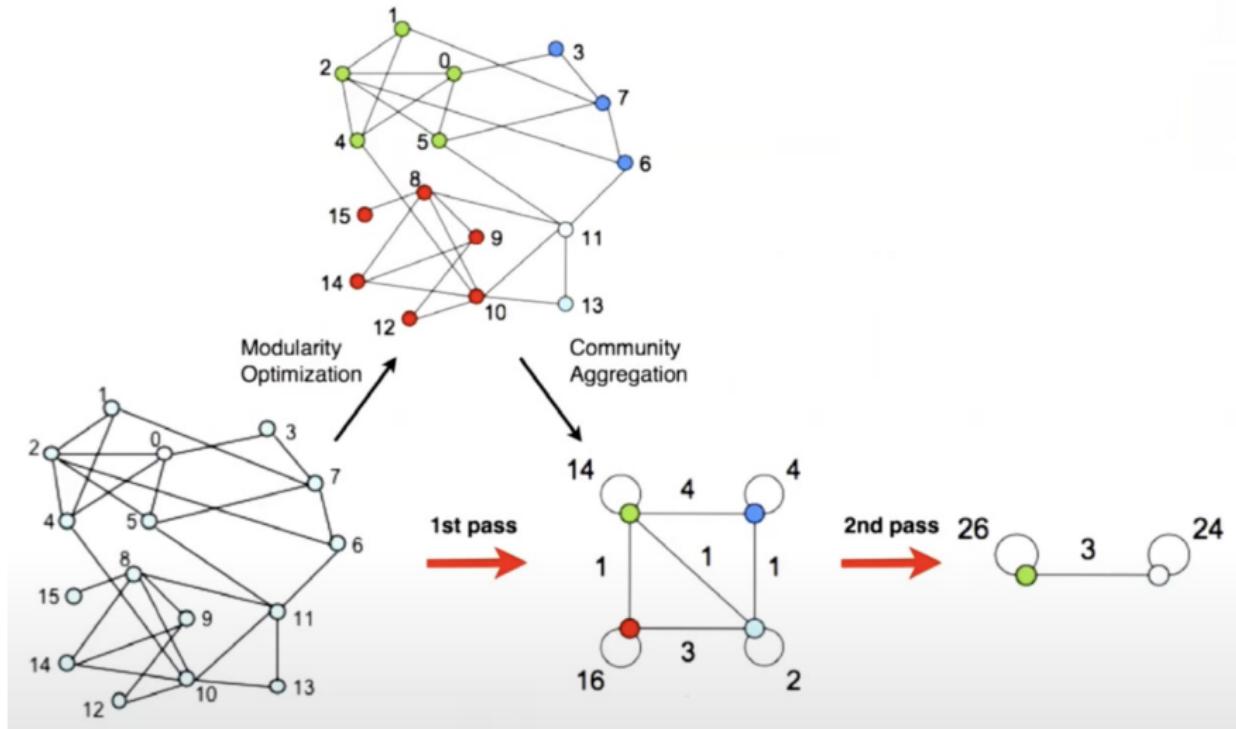
- Repeat the procedure sequentially to all nodes until no more improvement (local max of modularity).
  - The output of this phase depends on the order in which the nodes are considered -> research shows that this doesn't significantly affect the overall modularity.



# Community Detection: Louvain method

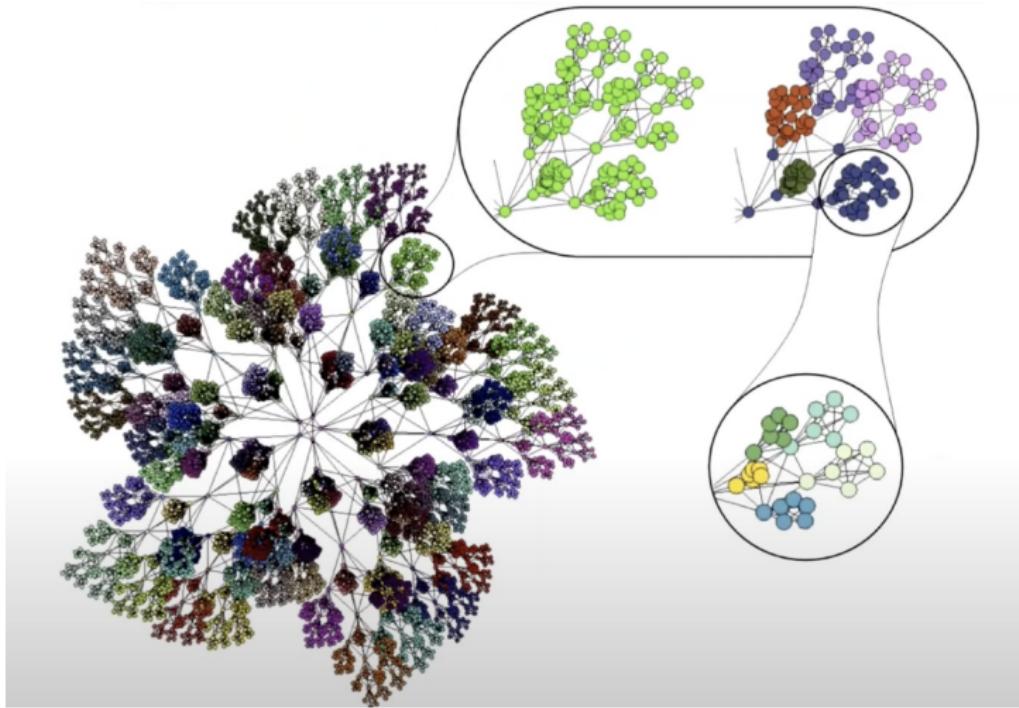
- ✓ Phase II
  - Nodes from communities grouped into super nodes
  - Links between nodes of the same community  $c$  are represented by self-loops weighted by adding up the links between these nodes:  
 $c$  has a loop edge with weight  $\sum_{v_i, v_j \in c} A_{i,j}$
  - Links between communities weighted by adding up the links between community's nodes each  $(c_I, c_J)$  has a link with weight  $\sum_{v_i \in C_I, v_j \in C_J} A_{i,j}$

# Community Detection: Louvain Algorithm



# Community Detection: Louvain Algorithm

- ✓ This figure shows many local nested communities to be easily explored



# Community Detection: Walktrap Approach

- ✓ Based on the random walk (Pons and Latapy, 2006) ↗
- ✓ Consider random walk on graph at each time step walk moves to neighbors randomly uniformly

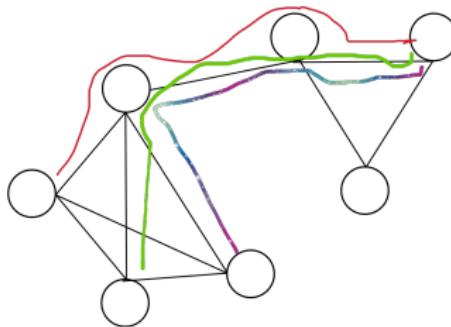
$$P_{ij} = \frac{A_{ij}}{d_i} \quad P = D^{-1}A \quad D_{ii} = \text{diag}(d_i)$$

$P_{ij}^t$  the probability to get from  $i$  to  $j$  in  $t$  steps

- ✓ The key idea:

- Two nodes  $i$  and  $j$  in the same community have a high  $P_{ij}^t$
- Two nodes  $i$  and  $j$  in the same community have similar  $P_{ik}^t \approx P_{kj}^t$
- Distance (similarity) between two nodes;

$$r_{ij}(t) = \sqrt{\sum_{k=1}^n \frac{(P_{ik}^t - P_{jk}^t)^2}{d_k}} = ||D^{-\frac{1}{2}} P_i^t - D^{-\frac{1}{2}} P_j^t||$$



# Community Detection: Walktrap Approach

- ✓ Based on the random walk
- ✓ Consider random walk on graph, at each time step walk moves to neighbors randomly uniformly

$$P_{ij} = \frac{A_{ij}}{d_i} \quad P = D^{-1} A \quad D_{ii} = \text{diag}(d_i)$$

$P_{ij}^t$  the probability to get from  $i$  to  $j$  in  $t$  steps

- ✓ The key idea:

- Two nodes  $i$  and  $j$  in the same community have a high  $P_{ij}^t$
- Two nodes  $i$  and  $j$  in the same community have similar  $P_{ik}^t \approx P_{kj}^t$
- Distance (similarity) between two nodes;

$$r_{ij}(t) = \sqrt{\sum_{k=1}^n \frac{(P_{ik}^t - P_{jk}^t)^2}{d_k}} = ||D^{-\frac{1}{2}} P_i^t - D^{-\frac{1}{2}} P_j^t||$$

- ✓ Computation is approximated by:

$$P_{ik}^t \approx \frac{N_{ik}}{N_i}$$

$N_{ik}$  the number of times to end up on node  $k$  starting from  $i$  and  $N_i$  is the number of walks starting from  $i$

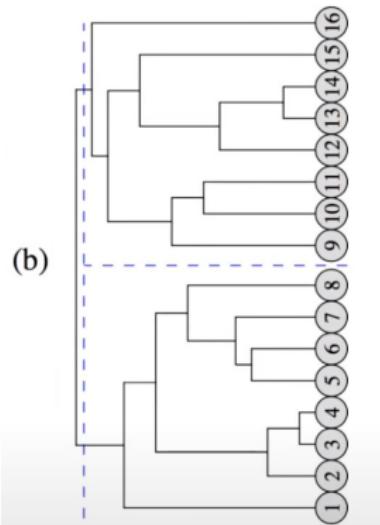
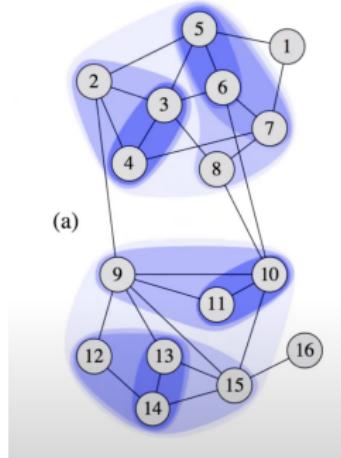


# Community Detection: Walktrap Approach

- ✓ Assign to each node its own community
- ✓ Compute the distance between adjacent nodes ( $r_{ij}(t)$ )
- ✓ Choose the two closest communities and merge them
- ✓ Update distance between communities

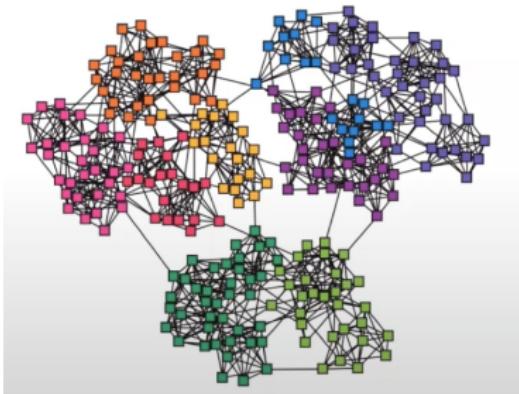
$$r_{C_1 C_2}(t) = \sqrt{\sum_{k=1}^n \frac{(P_{C_1 k}^t - P_{C_2 k}^t)^2}{d_k}} = ||D^{-\frac{1}{2}} P_{C_1}^t - D^{-\frac{1}{2}} P_{C_2}^t|| \quad P_{C_i k}^t = \frac{1}{|C_i|} \sum_{j \in C_i} P_{jk}^t$$

- ✓ After  $n$  steps finish with one community



# Community Detection: Girvan-Newman Approach

- ✓ Edge betweenness of  $e$  is the fraction of the number of shortest paths between all nodes  $s$  and  $t$  going through the edge  $e$ :  $ebt_e = \sum_{s \neq t} \frac{n_{s,t}^e}{n_{s,t}}$
- ✓ Focus on edges that connect communities. Construct communities by progressively removing edges with the highest edge betweenness value.
  - For all  $e \in E$  compute  $ebt_e$ , remove edge  $e$  with the largest  $ebt_e$
  - Repeat until all edges left. Stop when it splits in 2 components. The output is a dendrogram.



# Community Detection: Girvan-Newman Approach

- ✓ Community Quality: the higher modularity score the better is the community

