Class 10:
Defining a community on a local level

Course: Computational Network Analysis

Prof. Dr. Claudia Müller-Birn, Alexa Schlegel

Institute of Computer Science, «Human-Centered Computing»

Mar 4, 2016

Recap

- You understood the difference between the discrete approach and the continuous approach of describing temporal networks
- You know what parameter you can use to define the size of the time window.
- You learnt about the time-respecting path in a network.
- You can discuss different measures that consider the time-respective path in temporal networks.

Today’s outline

Defining a community

Self-referring definition of a community

Describing overlapping communities with the clique percolation method

Understanding community evolution based on the clique percolation method

Definition of Community

(Online) Community

- „Virtual communities are social aggregations that emerge from the Net when enough people carry on those public discussions long enough, with sufficient human feeling, to form webs of personal relationships in cyberspace.“
- There are two arguments for detecting groups in networks
  - application argument
  - structural argument

Rheingold, Howard: The virtual community: homesteading on the electronic frontier. Reading, Mass. : Addison-Wesley Publishing, 1993

Application oriented perspective

- **Explicit** user-defined communities
  - In social network services, e.g. Linked-In, Facebook
  - In communication services, e.g. mailing lists, comments in Stackoverflow

- **Implicit** communities formed by affiliation
  - In peer production communities, e.g. Wikidata, Stackoverflow
  - Using the same web services, e.g. Yahoo

Why might it be necessary to extract groups?

- Starting point
  - Not all sites provide a community platform
  - Not all people want to make effort to join groups
  - Groups can change dynamically

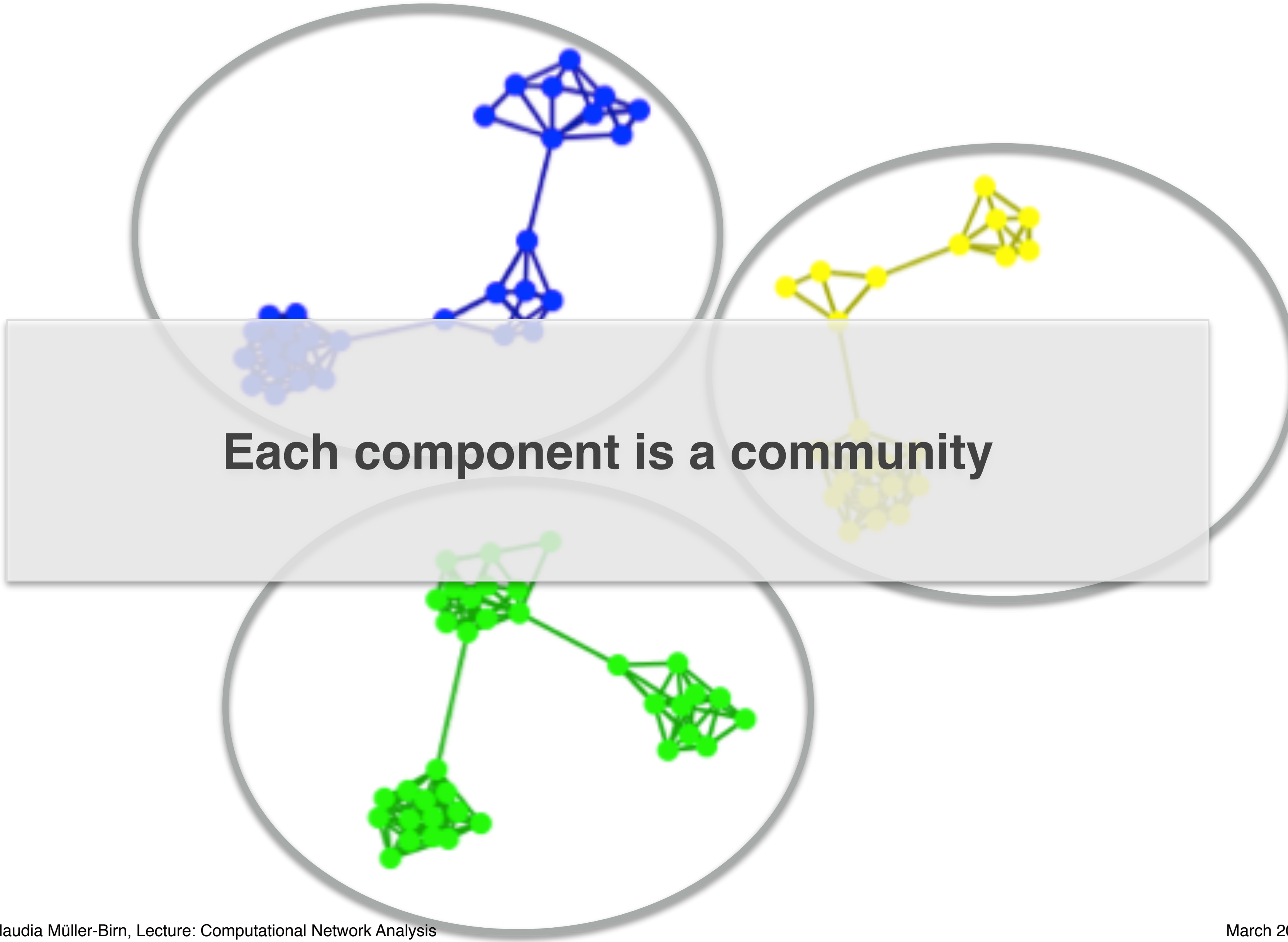
- However, research has shown that
  - the feeling of belonging to a group increases retention
  - people in groups you have greater uniformity in their opinions

- Networks provides rich information about the relationship between users
  - Can complement other kinds of information, e.g. user profile
  - Provide basic information for social computing tasks, such as recommendation
  - Grouping of customers with similar interests
  - Determine threshold processes, such that:
    “I will adopt an innovation if some number of my contacts do.”
    “I will vote for a politician if a fraction of my contacts do.”

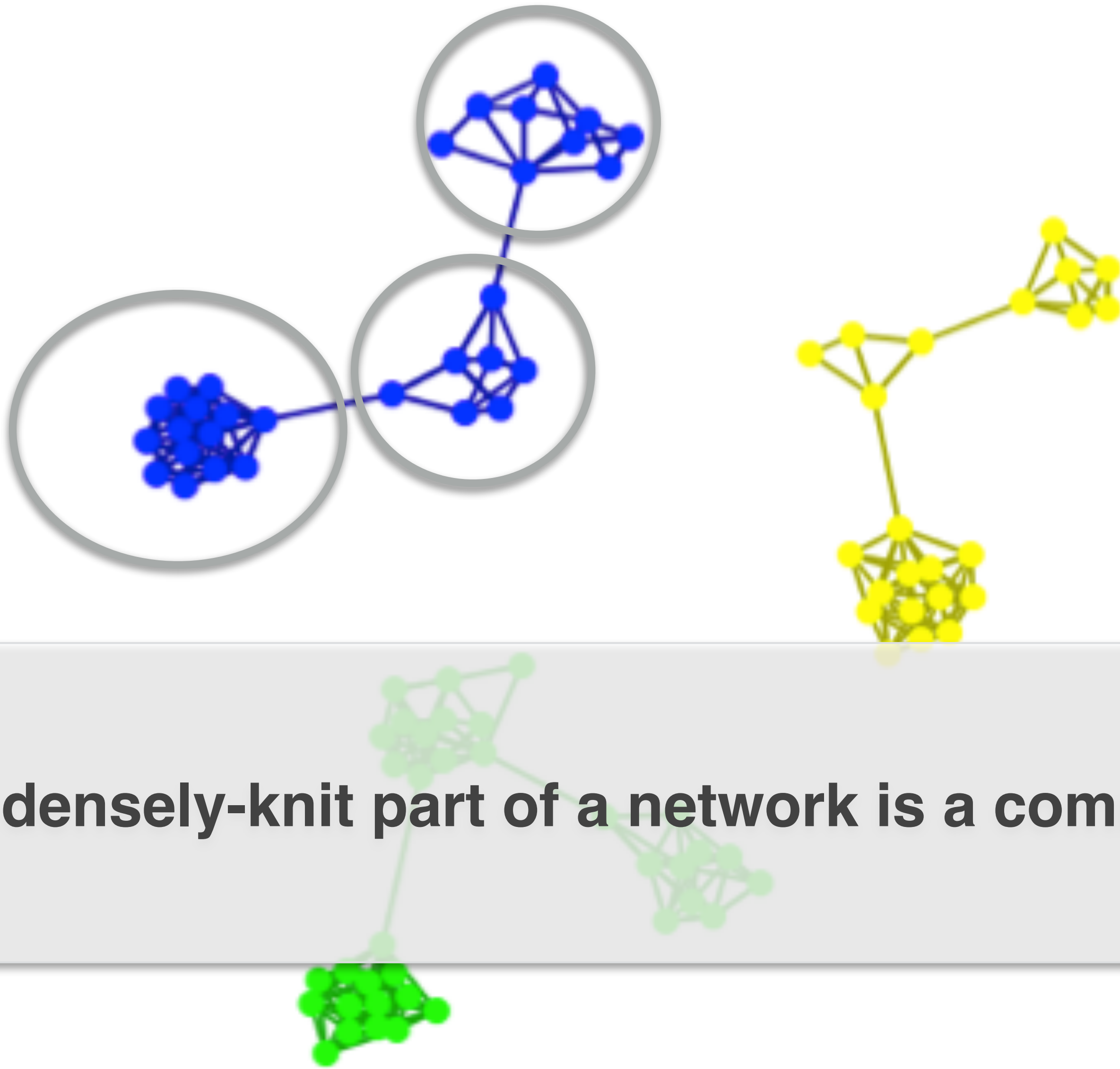
Structure-oriented perspective

- Networks are often structured; revealing the structure enables a better understanding of existing network dependencies
- Networks are increasingly complex and separation of networks into different parts might simplify their analysis

What is a community?



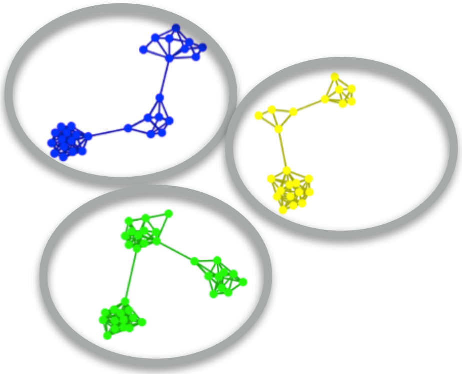
What is a community?

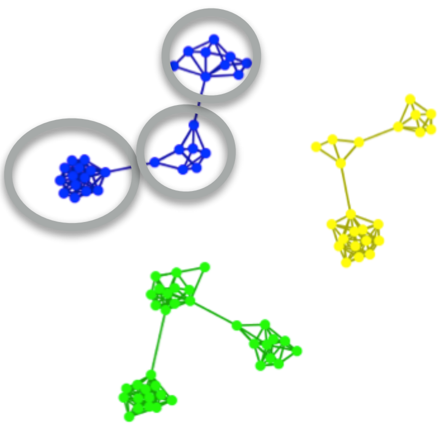


A densely-knit part of a network is a community

What is a community? Depends…

Network-level perspective





Node-level perspective

Node level perspective of a community

- Focus is on vertices of the subgraph under investigation and on its immediate neighborhood, disregarding the rest of the graph

- *Self-referring definition*
  - Subgraph alone is considered
  - For example: cliques, k-cliques, k-core, k-plex

- *Comparative definitions*
  - Mutual cohesion of the vertices in a subgraph is compared with their cohesion with external neighbors
  - For example: LS Set

Santo Fortunato, Claudio Castellano: Community Structure in Graphs. arXiv. http://arxiv.org/abs/0712.2716

How do we define a community?

- A community is following called a **cohesive subgroup** but can also be called groups, clusters, or modules in other contexts

- Cohesive subgroups is a subset of vertices among whom there are relatively strong, direct, intense, frequent, positives edges

- There are different lines of research
  - Scaling up methods to detect communities
  - Find hidden communities among heterogeneous interactions
  - Evolution of community membership

Characteristics of cohesive groups

- **Mutuality**: Group members choose each other to be included in the group. In a graph-theoretical sense, this means that they are adjacent.
- **Compactness**: Group members are well reachable for each other, though not necessarily adjacent. Graph-theoretically, this comes in two flavors: being well reachable can be interpreted as having short distances or high connectivity.
- **Density**: Group members have many contacts to each other. In terms of graph theory, that is group members have a large neighborhood inside the group.
- **Separation**: Group members have more contacts inside the group than outside.

Self-referring definition

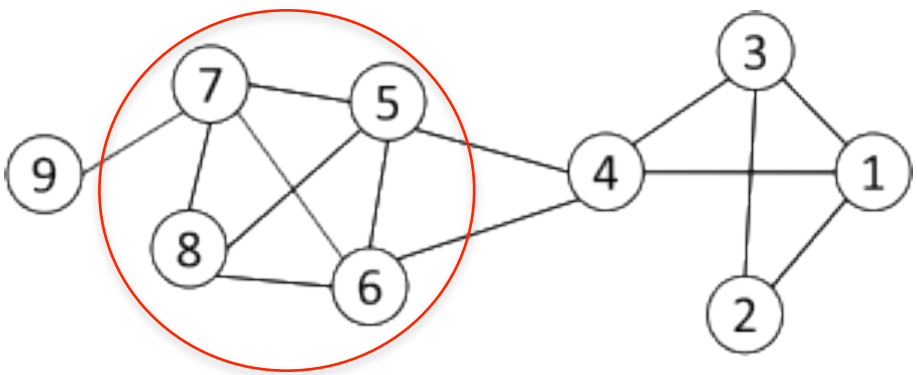
This chapter is mainly based on:

Wasserman, S, Faust, K. Social Network Analysis. Cambridge University Press. 1997.

Freie Universität Berlin

Clique

- Ideal maximum complete (sub-)graph, i.e. it is a set of vertices and all vertices are adjacent to each other
- A clique consists of at least three vertices and its density is 1
- Size of the clique is restricted to the degree k of vertices and therefore, the clique has not more than k+1 members
- Straightforward implementation to find cliques is very expensive in time complexity (NP-hard to find the maximum clique in a network)



{5,6,7,8}

Considerations in using cliques as subgroups

- Using the clique concept is
  - Not robust because one missing link can disqualify a clique
  - Not interesting because
    - Everybody is connected to everybody else
    - No core-periphery structure
    - No centrality measures apply

- Relaxations of the clique concept
  - **Reachability** of members, for example n-clique, n-clan
  - **Nodal degrees**, for example k-core, k-plex

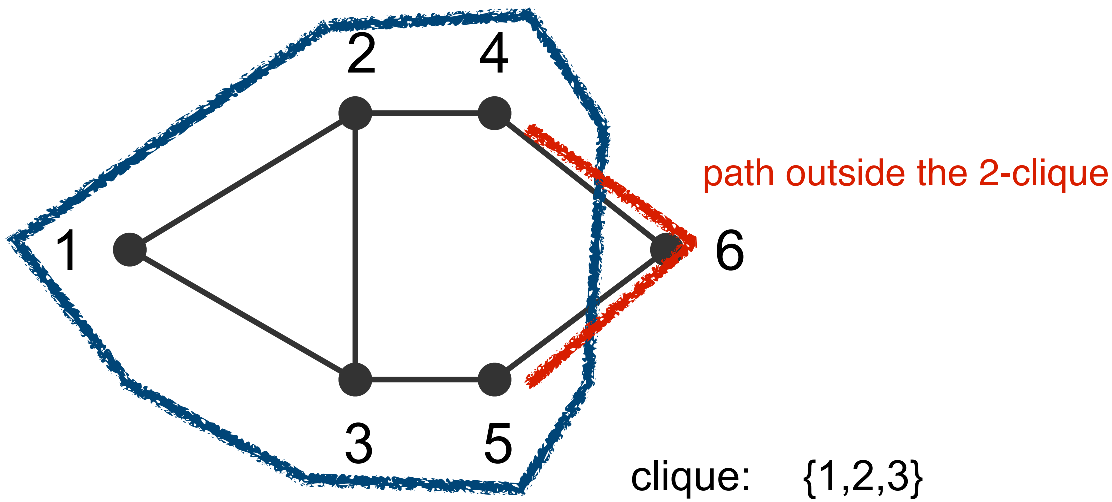
n-cliques

- An n-clique is a maximal subgraph in which the largest geodesic distance between any two vertices in the subgraph is no greater than k
- An n-clique is a subgraph with node set N_S such that

$$d(i, j) \leq k \text{ for all } n_i, n_j \in N_s$$

- When should you use this?
  - Members of the subgraph might not be adjacent but they are connected by a relatively short paths
  - Social processes take place through intermediaries in the investigated network

Example

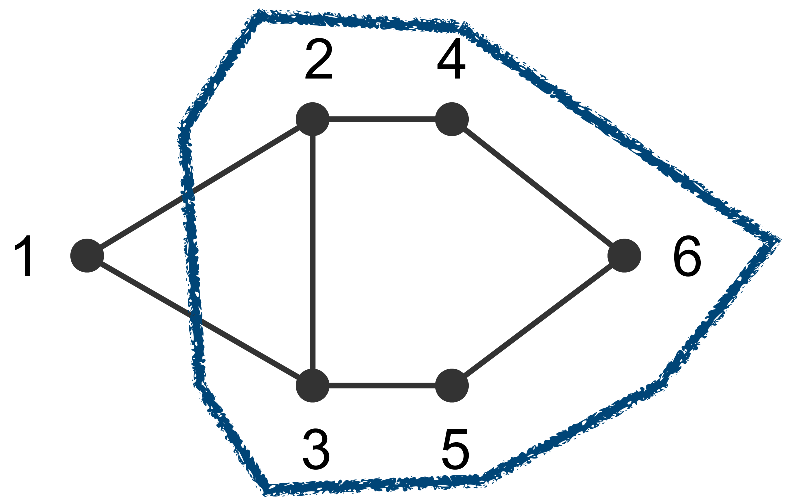


clique: {1,2,3}
2-clique: {1,2,3,4,5}, {2,3,4,5,6}

- Criticisms:
  - Diameter may be greater than n
  - n-clique may be disconnected (paths go through nodes not in subgroup)

n-clan

- Is a maximal complete subgraph, where each vertex has maximally the distance n in the resulting sub-graph



clique: {1,2,3}
2-clique: {1,2,3,4,5}, {2,3,4,5,6}
2-clan: {2,3,4,5,6}

Considerations in using cliques as subgroups

- Using the clique concept is
  - Not robust because one missing link can disqualify a clique
  - Not interesting because
    - Everybody is connected to everybody else
    - No core-periphery structure
    - No centrality measures apply

- Relaxations of the clique concept
  - **Reachability** of members, for example n-clique, n-clan
  - **Nodal degrees**, for example k-core, k-plex

k-core

- Cohesive subgroup based on the nodal degree

- A k-core is a maximal subgraph, in which each vertex is adjacent to at least a minimum number, k, of the other vertices in the subgraph

- A subgraph N_S is a k-core, if

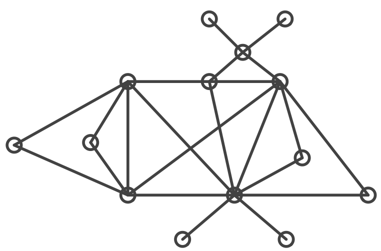
$$d_s(i) \geq k \text{ for all } n_i \in N_S$$

where $d_s(i)$ is the degree of a node i within a subgraph

- By varying the value of k (that is, how many members of the group do you have to be connected to), different pictures can emerge

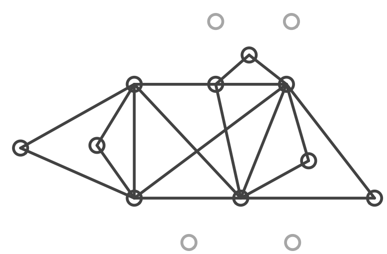
Example of calculating a 3-core

1) Start with the whole network
2) Set k=2
3) Remove all vertices with a degree less than k (check in every iteration all vertices)
4) k=k+1, go to step 3.

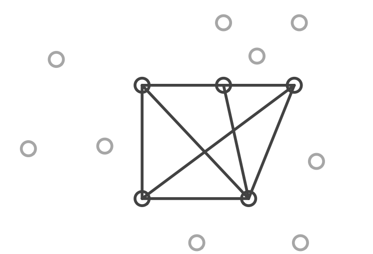


Result: k-core or set of k-cores, since each vertex is connected to at least k others

2-Core
(all vertices with degree < 2 are deleted)



3-Core
(all vertices with degree < 3 are deleted)



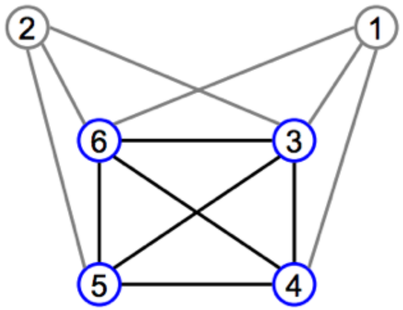
k-plex

- Cohesive subgroup based on the adjacency of subgroup members
- A k-plex is a maximal subgraph containing a number of g_s vertices in which each vertex is adjacent to no fewer than g_s-k vertices in the subgraph or each vertex in G_S has at most k non-neighbors in the subgraph
- The degree of a vertex in a subgraph is $d_s(i)$
- Thus, a k-plex is a subgraph in which

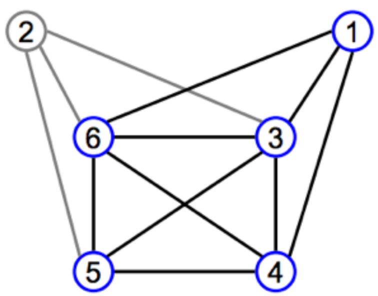
$$d_s(i) \geq (g_s - k) \text{ for all } n_i \in N_S$$

- k-plex is more robust than a k-clique and the removal of a single vertex is less likely to leave the subgraph disconnected

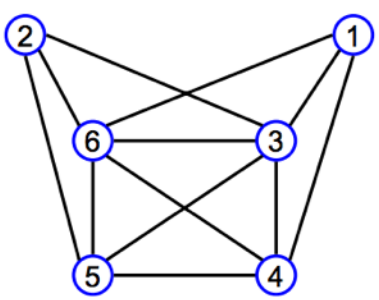
Example k-plex



- {3,4,5,6} is a 1-plex ... the "regular" clique



- {1,3,4,5,6} is a 2-plex (and NOT a 1-plex)

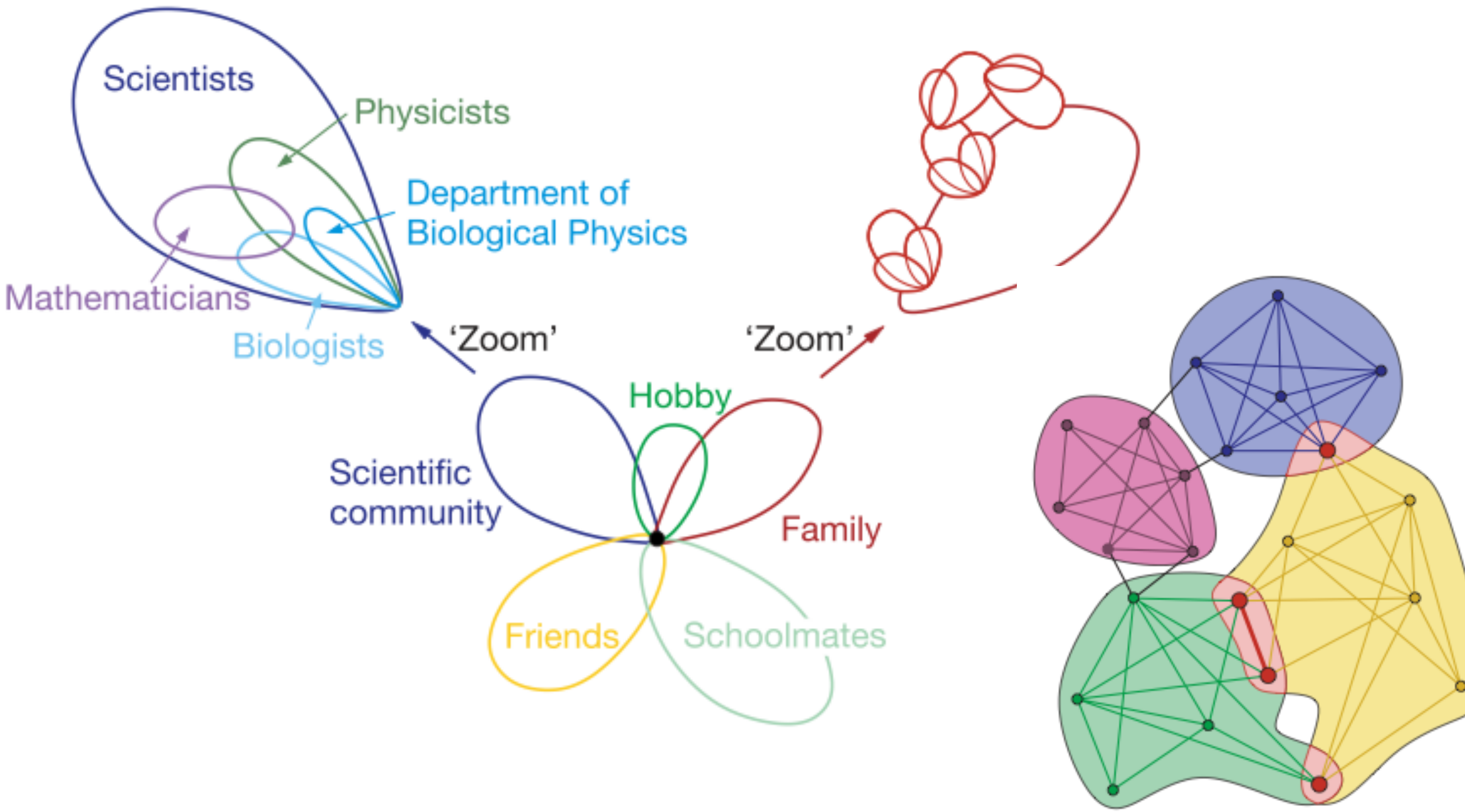


- {1,2,3,4,5,6} is a 3-plex (and NOT a 2-plex)

- <u>Please note:</u>
  - Choosing k is difficult so meaningful results can be found
  - one should look at resulting group sizes - they should be larger then k by some margin

Clique percolation method (CPM)

Overlapping communities



(Palla et al. 2005 and 2007)
(Fakas et al. 2007)

Palla, Gergely, Albert-László Barabási, and Tamás Vicsek. Quantifying social group evolution. Nature 446.7136 (2007): 664-667.

Procedure of detecting percolation communities

Step 1:
Locate all complete subgraphs, i.e. k-cliques, that are not part of a larger subgraph

Step 2:
Identify communities based on clique-clique overlap matrix

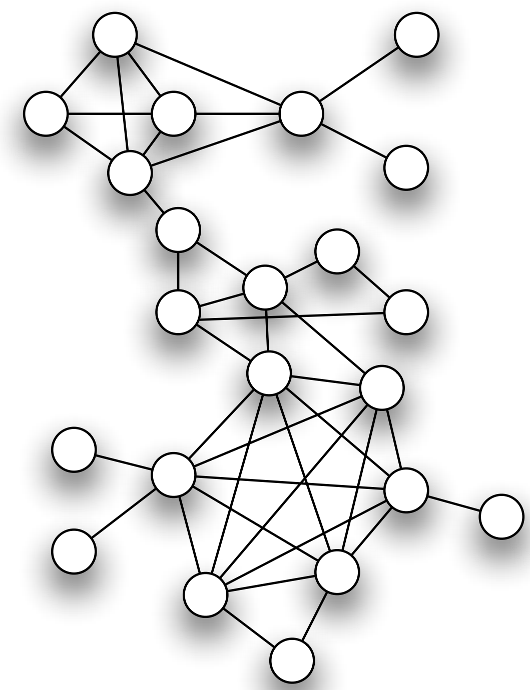
Step 3:
Specify “optimal” percolation community structure

(Palla et al. 2005 and 2007)
(Fakas et al. 2007)

Step 1: Locate all complete subgraphs

1. Compute the degree sequence for G
2. Choose k=min(r_v) (corresponds to size of clique)
3. Select v (if d(v)≧k) and assign it to set A, otherwise proceed with step 6
4. Create a disjunct set *B* that contains all vertices that are adjacent to v
5. Brute Force Approach:
   1. Choose the vertex u (decreasing/increasing order of their indices) in B and move it to A
   2. Remove all vertices in B that are not adjacent to u
   3. Proceed with step i. and ii. until B = ∅
   4. Check set A: If IAI = k then a new clique is found; otherwise proceed with step i.
   5. If all combinations are checked; remove vertex u and its edges and proceed with step 3
   6. If G = ∅ proceed with step 6
6. Decrease clique size by k = k-1
7. Go to step 3

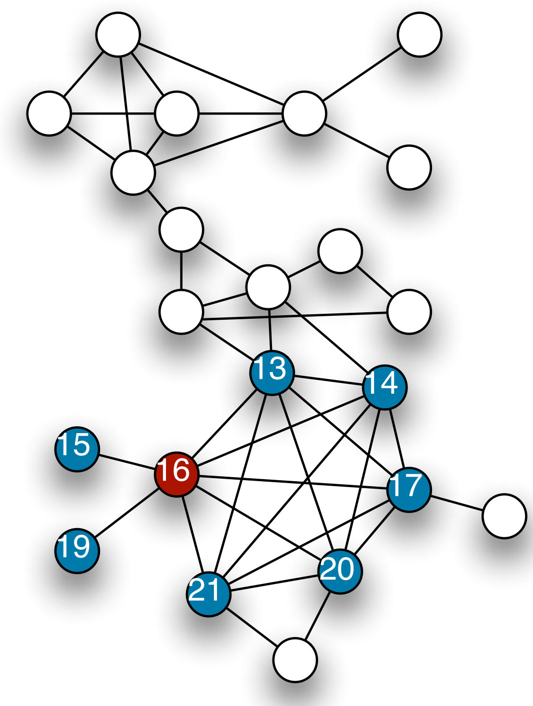
k = 7



Step 1: Locate all complete subgraphs

1. Compute the degree sequence for G
2. Choose k=min(r_v) (corresponds to size of clique)
3. Select v (if d(v)≧k) and assign it to set A, otherwise proceed with step 6
4. Create a disjunct set *B* that contains all vertices that are adjacent to v
5. Brute Force Approach:
    1. Choose the vertex u (decreasing/increasing order of their indices) in B and move it to A
    2. Remove all vertices in B that are not adjacent to u
    3. Proceed with step i. and ii. until B = ∅
    4. Check set A: If IAI = k then a new clique is found; otherwise proceed with step i.
    5. If all combinations are checked; remove vertex u and its edges and proceed with step 3
    6. If G = ∅ proceed with step 6
6. Decrease clique size by k = k-1
7. Go to step 3

k = 7

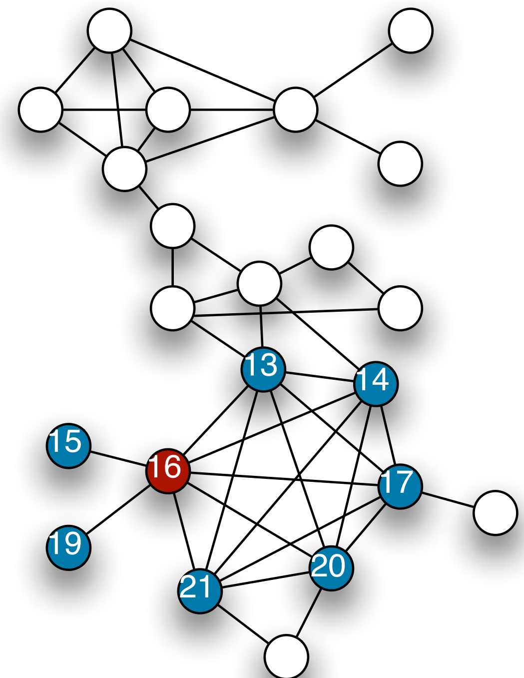


A = {16}
B = {13,14,15,17,19,20,21}

Step 1: Locate all complete subgraphs

1. Compute the degree sequence for G
2. Choose k=min(r_v) (corresponds to size of clique)
3. Select v (if d(v)≧k) and assign it to set A, otherwise proceed with step 6
4. Create a disjunct set *B* that contains all vertices that are adjacent to v
5. Brute Force Approach:
    1. Choose the vertex u (decreasing/increasing order of their indices) in B and move it to A
    2. Remove all vertices in B that are not adjacent to u
    3. Proceed with step i. and ii. until B = ∅
    4. Check set A: If IAI = k then a new clique is found; otherwise proceed with step i.
    5. If all combinations are checked; remove vertex u and its edges and proceed with step 3
    6. If G = ∅ proceed with step 6
6. Decrease clique size by k = k-1
7. Go to step 3

k = 7

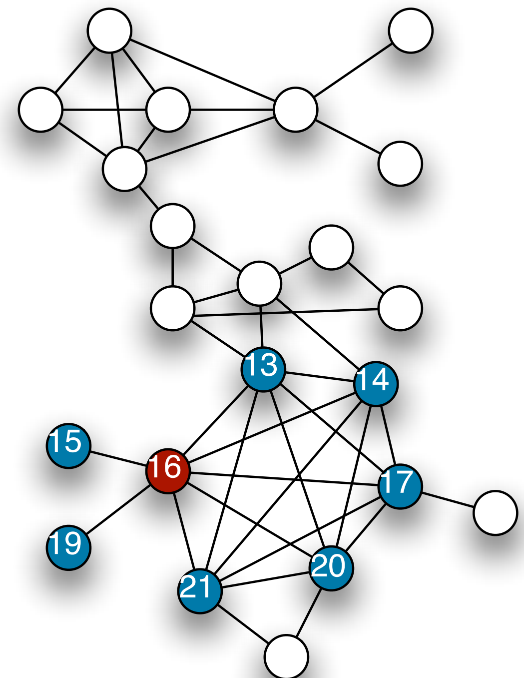


A = {16}
B = {13,14,15,17,19,20,21}

Step 1: Locate all complete subgraphs

1. Compute the degree sequence for G
2. Choose k=min(r_v) (corresponds to size of clique)
3. Select v (if d(v)≧k) and assign it to set A, otherwise proceed with step 6
4. Create a disjunct set *B* that contains all vertices that are adjacent to v
5. Brute Force Approach:
    1. Choose the vertex u (decreasing/increasing order of their indices) in B and move it to A
    2. Remove all vertices in B that are not adjacent to u
    3. Proceed with step i. and ii. until B = ∅
    4. Check set A: If IAI = k then a new clique is found; otherwise proceed with step i.
    5. If all combinations are checked; remove vertex u and its edges and proceed with step 3
    6. If G = ∅ proceed with step 6
6. Decrease clique size by k = k-1
7. Go to step 3

k = 7

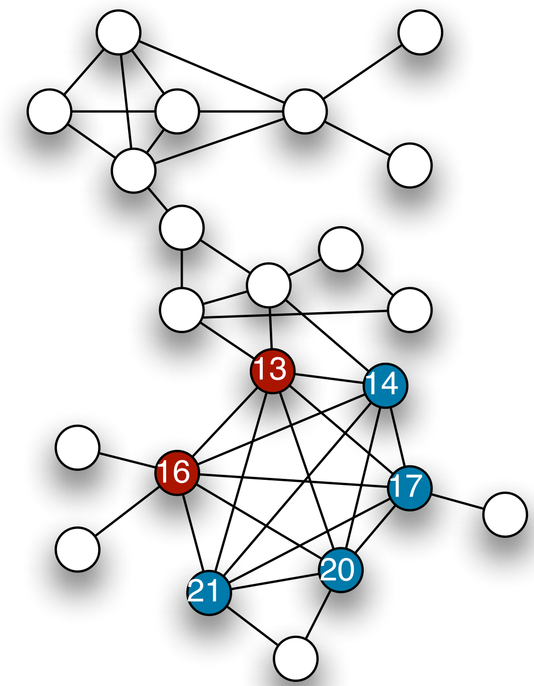


A = {16, ~~15~~, 19}
B = {13,14,17,20,21}

Step 1: Locate all complete subgraphs

1. Compute the degree sequence for G
2. Choose k=min(r_v) (corresponds to size of clique)
3. Select v (if d(v)≧k) and assign it to set A, otherwise proceed with step 6
4. Create a disjunct set *B* that contains all vertices that are adjacent to v
5. Brute Force Approach:
    1. Choose the vertex u (decreasing/increasing order of their indices) in B and move it to A
    2. Remove all vertices in B that are not adjacent to u
    3. Proceed with step i. and ii. until B = ∅
    4. Check set A: If IAI = k then a new clique is found; otherwise proceed with step i.
    5. If all combinations are checked; remove vertex u and its edges and proceed with step 3
    6. If G = ∅ proceed with step 6
6. Decrease clique size by k = k-1
7. Go to step 3

k = 7

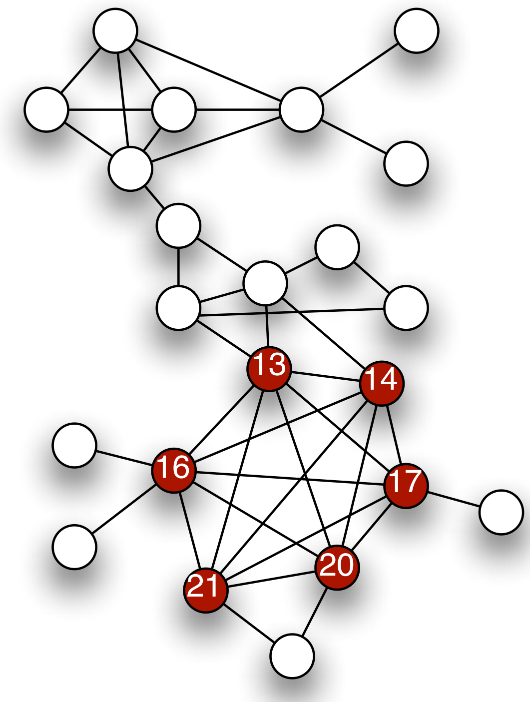


A = {16,~~19,~~ 13}
B = {14,17,20,21}

Step 1: Locate all complete subgraphs

1. Compute the degree sequence for G
2. Choose k=min(r_v) (corresponds to size of clique)
3. Select v (if d(v)≧k) and assign it to set A, otherwise proceed with step 6
4. Create a disjunct set *B* that contains all vertices that are adjacent to v
5. Brute Force Approach:
    1. Choose the vertex u (decreasing/increasing order of their indices) in B and move it to A
    2. Remove all vertices in B that are not adjacent to u
    3. Proceed with step i. and ii. until B = ∅
    4. Check set A: If IAI = k then a new clique is found; otherwise proceed with step i.
    5. If all combinations are checked; remove vertex u and its edges and proceed with step 3
    6. If G = ∅ proceed with step 6
6. Decrease clique size by k = k-1
7. Go to step 3

k = 7

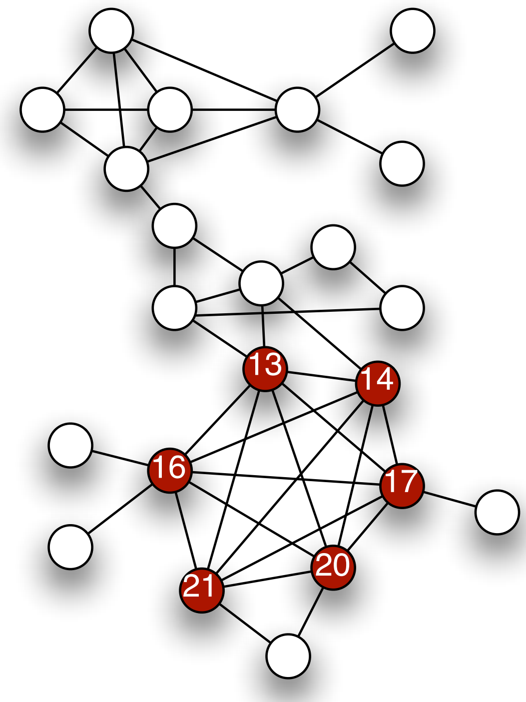


A = {13,14,16,17,20,21}
B = ∅

Step 1: Locate all complete subgraphs

1. Compute the degree sequence for G
2. Choose k=min(r_v) (corresponds to size of clique)
3. Select v (if d(v)≧k) and assign it to set A, otherwise proceed with step 6
4. Create a disjunct set *B* that contains all vertices that are adjacent to v
5. Brute Force Approach:
    1. Choose the vertex u (decreasing/increasing order of their indices) in B and move it to A
    2. Remove all vertices in B that are not adjacent to u
    3. Proceed with step i. and ii. until B = ∅
    4. Check set A: If IAI = k then a new clique is found; otherwise proceed with step i.
    5. If all combinations are checked; remove vertex u and its edges and proceed with step 3
    6. If G = ∅ proceed with step 6
6. Decrease clique size by k = k-1
7. Go to step 3

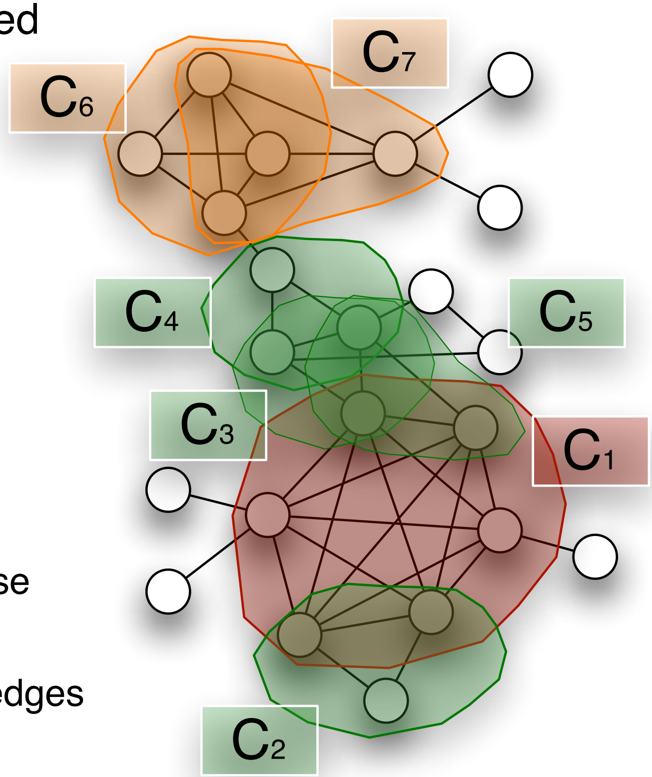
k = 7



A = {13,14,16,17,20,21}
B = ∅

Step 1: Locate all complete subgraphs

1. Compute the degree sequence for G
2. Choose $k=\min(r_v)$ (corresponds to size of clique)
3. Select v (if $d(v) \geqq k$) and assign it to set A, otherwise proceed with step 6
4. Create a disjunct set *B* that contains all vertices that are adjacent to v
5. Brute Force Approach:
    1. Choose the vertex u (decreasing/increasing order of their indices) in B and move it to A
    2. Remove all vertices in B that are not adjacent to u
    3. Proceed with step i. and ii. until B = ∅
    4. Check set A: If IAI = k then a new clique is found; otherwise proceed with step i.
    5. If all combinations are checked; remove vertex u and its edges and proceed with step 3
    6. If G = ∅ proceed with step 6
6. Decrease clique size by k = k-1
7. Go to step 3



Procedure of detecting percolation communities

Step 1:
Locate all complete subgraphs, i.e. k-cliques, that are not part of a larger subgraph

Step 2:
Identify communities based on clique-clique overlap matrix

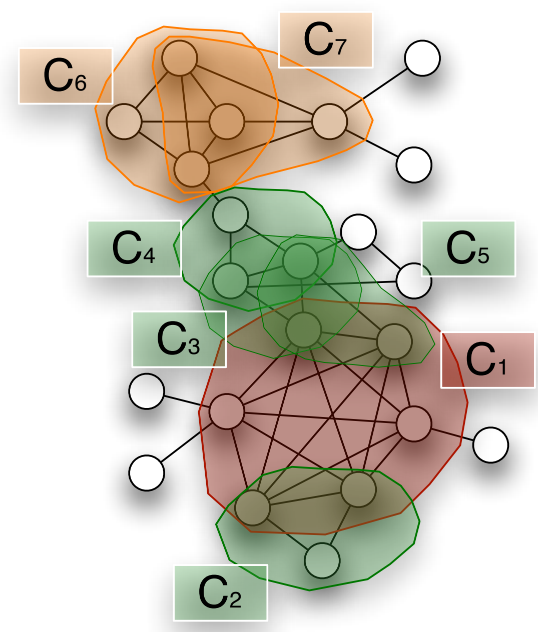
Step 3:
Specify "optimal" percolation community structure

(Palla et al. 2005 and 2007)
(Fakas et al. 2007)

Step 2: Identify communities

1. Construct a clique-clique overlap matrix
2. Define k (start with max(r_v))
3. Replace all off-diagonal entries smaller than (k-1) by zero
4. Replace every diagonal element smaller than k by zero
5. Replace all remaining entries by 1
6. Reduce k by 1 and go to step 3

	C_1	C_2	C_3	C_4	C_5	C_6	C_7
C_1	6	2	1	0	2	0	0
C_2	2	3	0	0	0	0	0
C_3	1	0	3	2	2	0	0
C_4	0	0	2	3	1	0	0
C_5	2	0	2	1	3	0	0
C_6	0	0	0	0	0	4	3
C_7	0	0	0	0	0	3	4



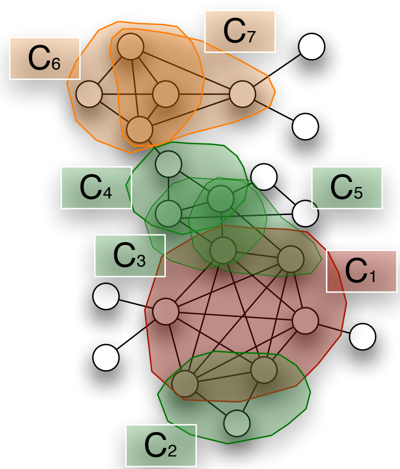
Step 2: Identify communities

shortcut
k = 3

1. Construct a clique-clique overlap matrix
2. Define k (start with max(r_v))
3. Replace all off-diagonal entries smaller than (k-1) by zero
4. Replace every diagonal element smaller than k by zero
5. Replace all remaining entries by 1
6. Reduce k by 1 and go to step 3

	C_1	C_2	C_3	C_4	C_5	C_6	C_7
C_1	6	2	1	0	2	0	0
C_2	2	3	0	0	0	0	0
C_3	1	0	3	2	2	0	0
C_4	0	0	2	3	1	0	0
C_5	2	0	2	1	3	0	0
C_6	0	0	0	0	0	4	3
C_7	0	0	0	0	0	3	4

	C_1	C_2	C_3	C_4	C_5	C_6	C_7
C_1	1	1	0	0	1	0	0
C_2	1	1	0	0	0	0	0
C_3	0	0	1	1	1	0	0
C_4	0	0	1	1	0	0	0
C_5	1	0	1	0	1	0	0
C_6	0	0	0	0	0	1	1
C_7	0	0	0	0	0	1	1



Procedure of detecting percolation communities

Step 1:
Locate all complete subgraphs, i.e. k-cliques, that are not part of a larger subgraph

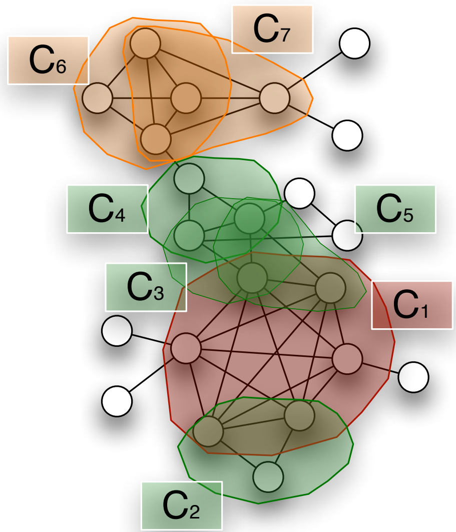
Step 2:
Identify communities based on clique-clique overlap matrix

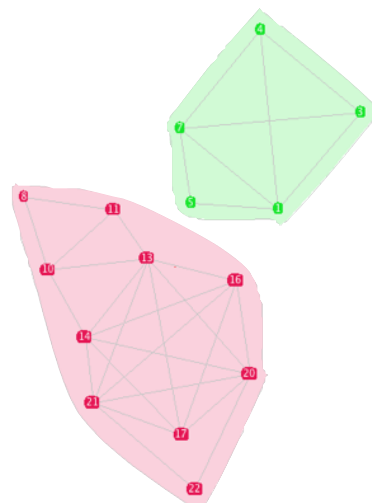
Step 3:
Specify "optimal" percolation community structure

(Palla et al. 2005 and 2007)
(Fakas et al. 2007)

Step 3: Specify size

k = 3



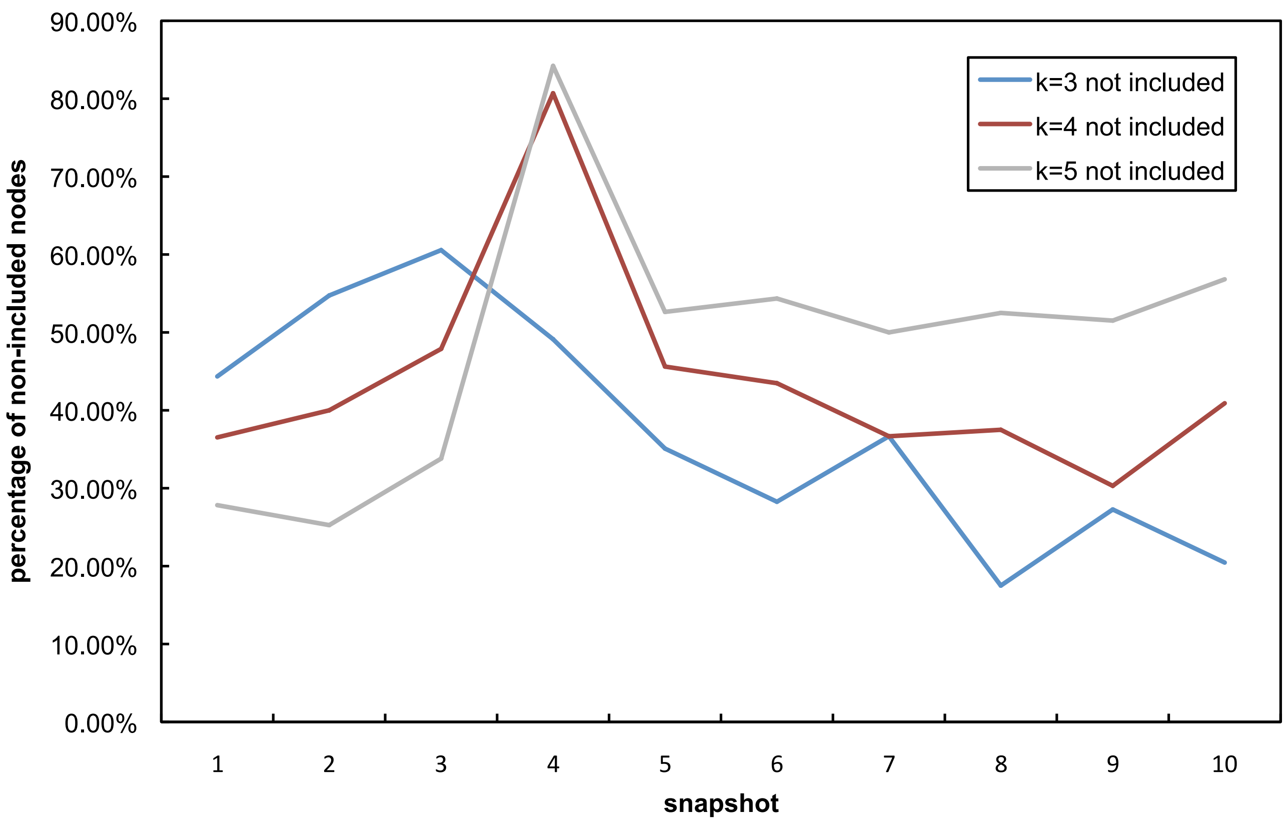


	C_1	C_2	C_3	C_4	C_5	C_6	C_7
C_1	6	2	1	0	2	0	0
C_2	2	3	0	0	0	0	0
C_3	1	0	3	2	2	0	0
C_4	0	0	2	3	1	0	0
C_5	2	0	2	1	3	0	0
C_6	0	0	0	0	0	4	3
C_7	0	0	0	0	0	3	4

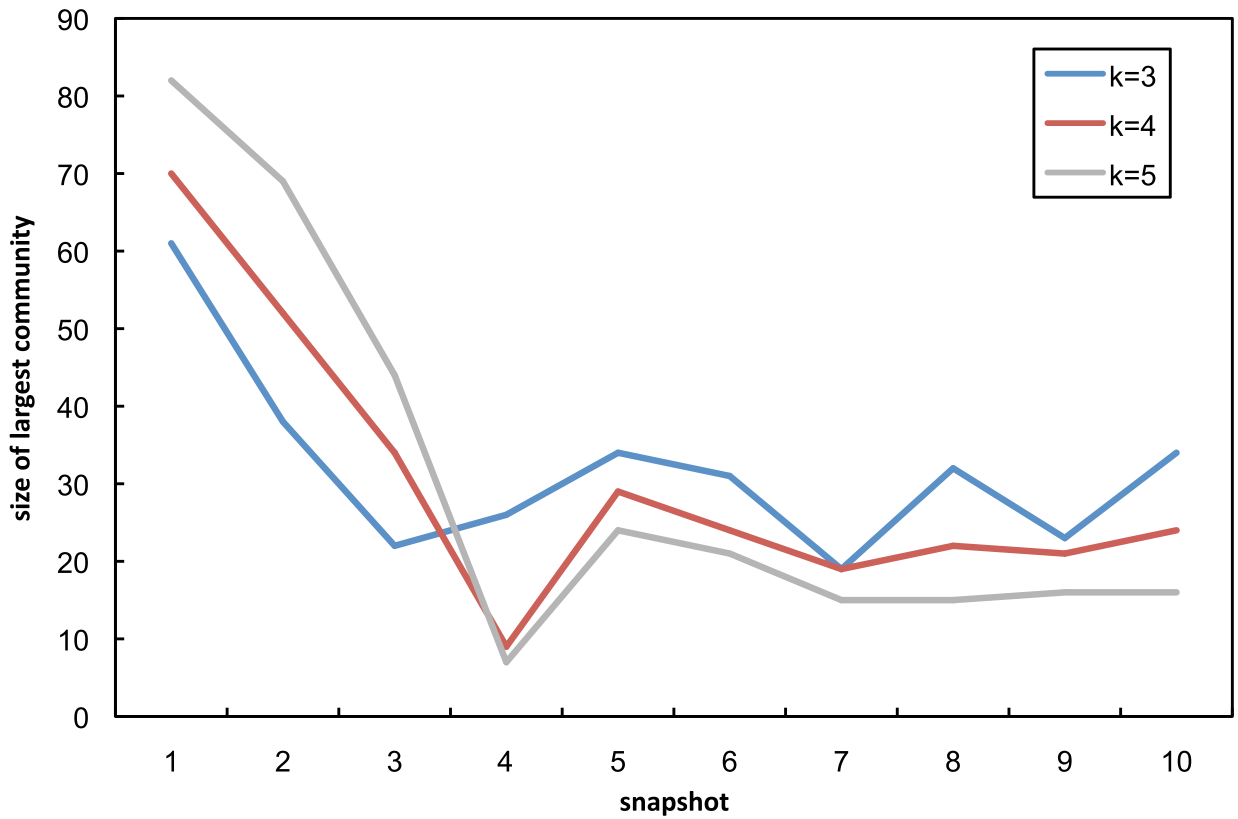
	C_1	C_2	C_3	C_4	C_5	C_6	C_7
C_1	1	1	0	0	1	0	0
C_2	1	1	0	0	0	0	0
C_3	0	0	1	1	1	0	0
C_4	0	0	1	1	0	0	0
C_5	1	0	1	0	1	0	0
C_6	0	0	0	0	0	1	1
C_7	0	0	0	0	0	1	1

block diagonal form

Specifying "best" structure

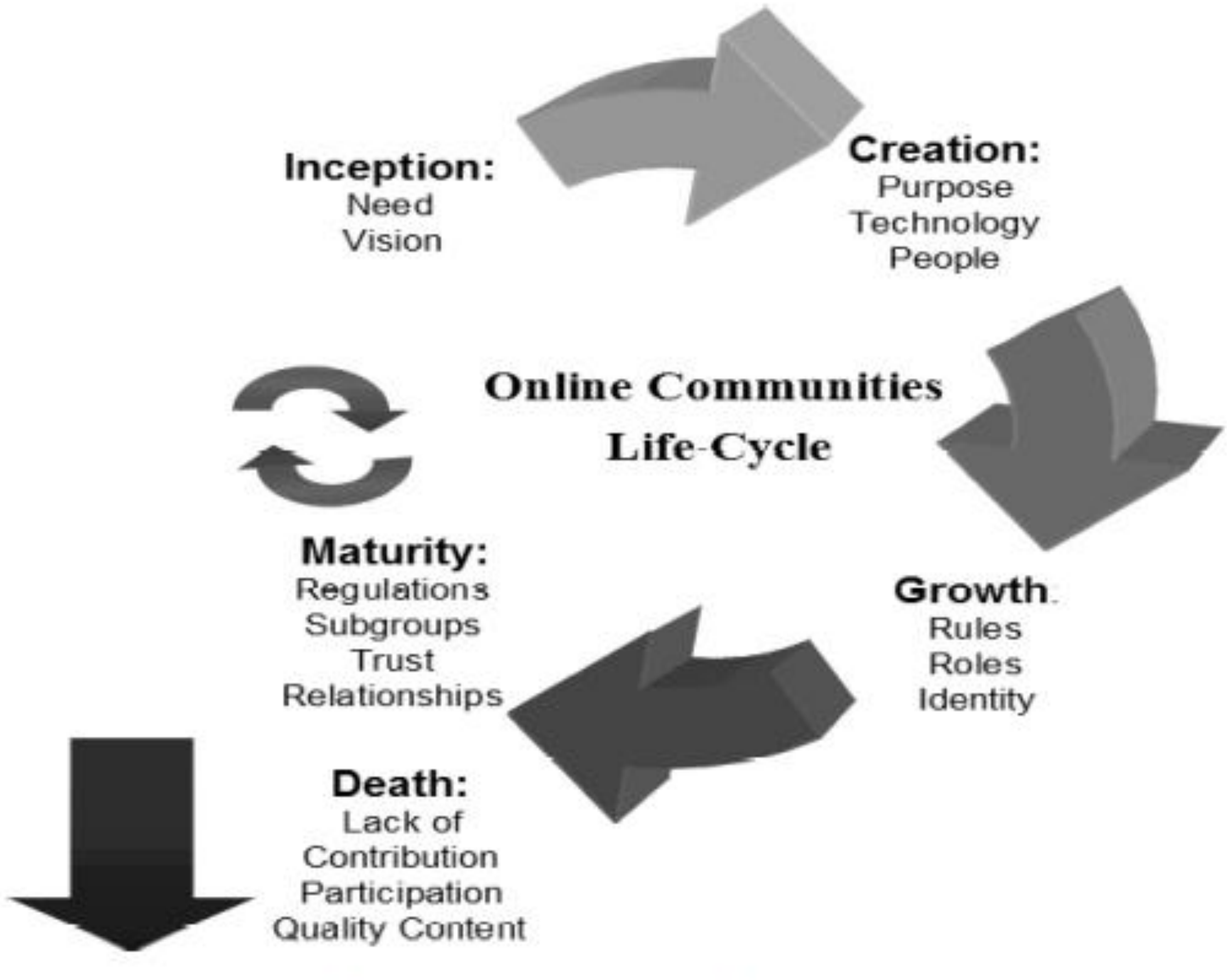


Specifying “best” structure (*cont.*)



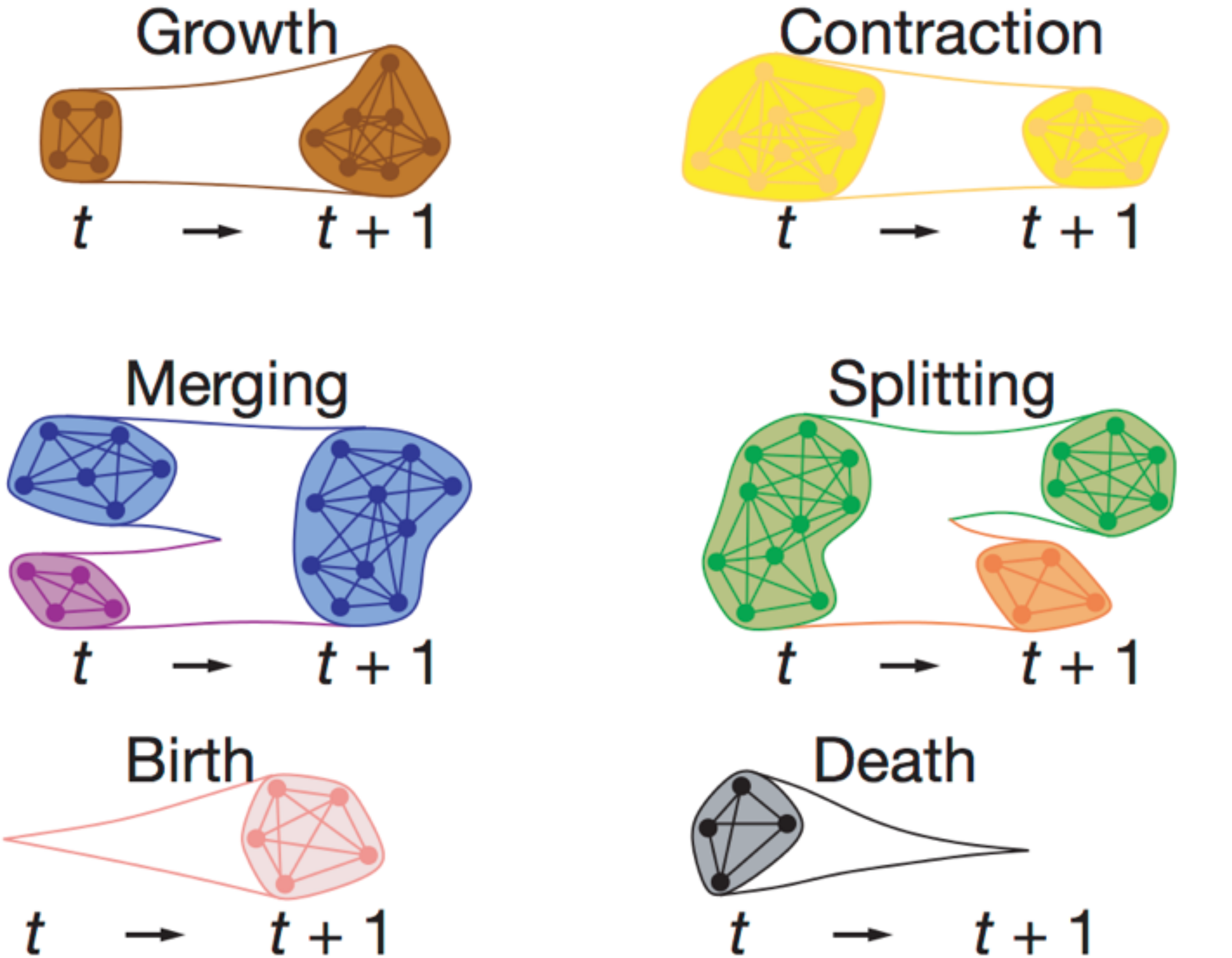
Community evolution based on CPM

The Online Community Lifecycle



Iriberri, A., & Leroy, G. (2009). A life-cycle perspective on online community success. ACM Comput. Surv., 41(2), 1–29.

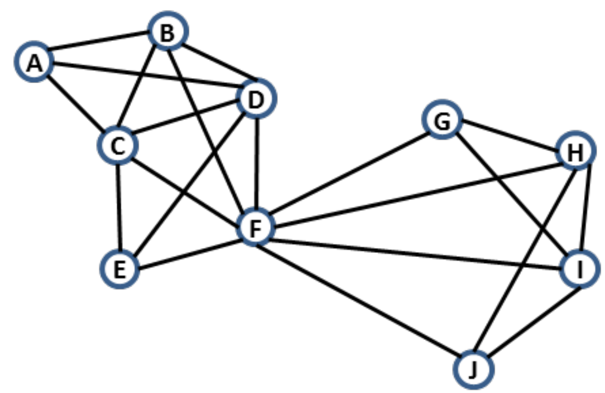
Community evolution



Palla, Gergely, Albert-László Barabási, and Tamás Vicsek. Quantifying social group evolution. Nature 446.7136 (2007): 664-667.

Algorithm - Community Matching [4]

	*Compute graph for each time slice **t***
1	Community detection for each graph
2	Matching detected communities for consecutive graphs
	2.1 Create joint graph
	2.2 Community detection for joint graph
	2.3 For each detected community **v** in joint graph - Find communities in **t** and **t+1** graph contained in **v** - Calculate relative overlap for each pair - Match communities in descending order
3	Post-processing



Algorithm - Community Matching

*Compute graph for each timestep **t***

1 Community detection for each graph

2 Matching detected communities for consecutive graphs

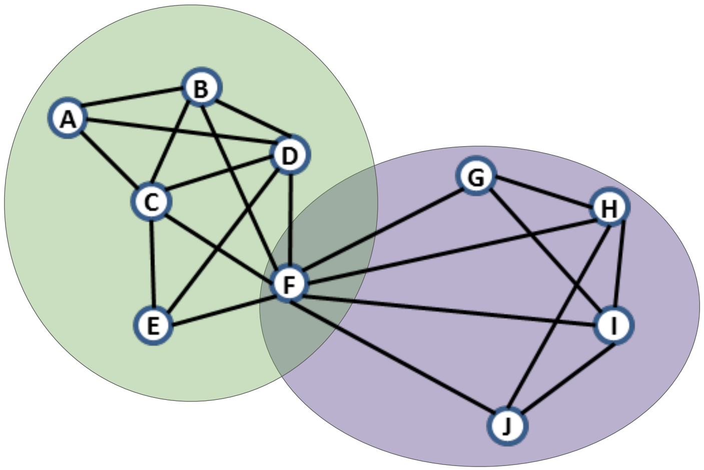
2.1 Create joint graph

2.2 Community detection for joint graph

2.3 For each detected community **v** in joint graph

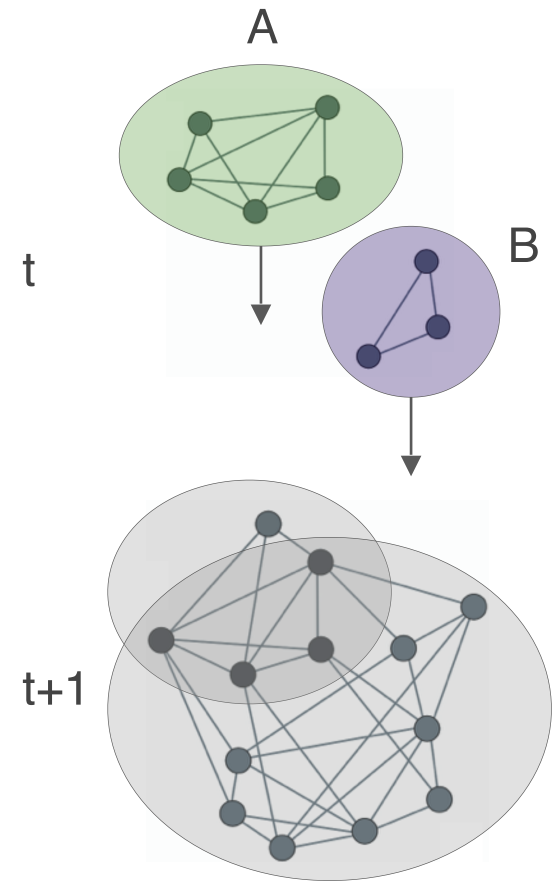
- Find communities in **t** and **t+1** graph contained in **v**
- Calculate relative overlap for each pair
- Match communities in descending order

3 Post-processing



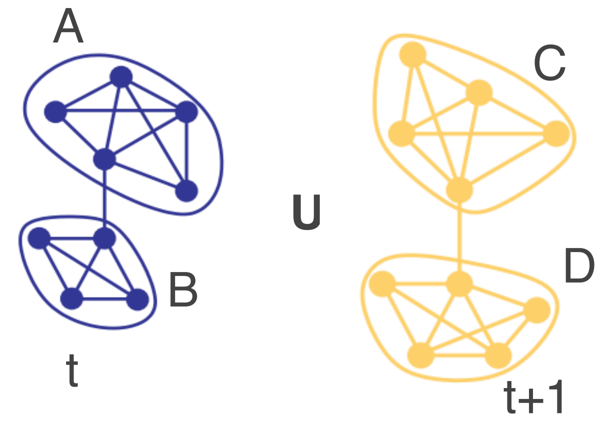
Algorithm - Community Matching

	*Compute graph for each timestep **t***
1	Community detection for each graph
2	Matching detected communities for consecutive graphs
	2.1 Create joint graph
	2.2 Community detection for joint graph
	2.3 For each detected community **v** in joint graph - Find communities in **t** and **t+1** graph contained in **v** - Calculate relative overlap for each pair - Match communities in descending order
3	Post-processing



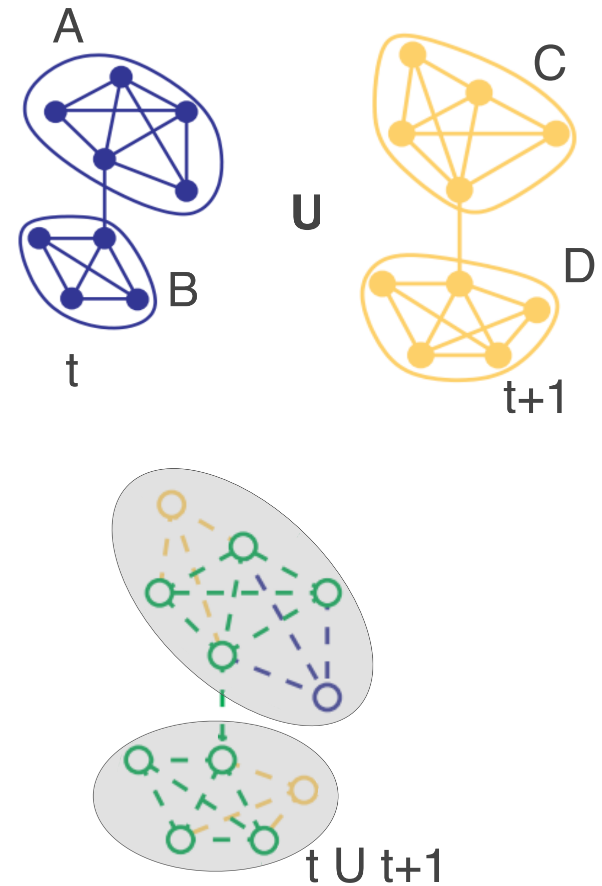
Algorithm - Community Matching

	*Compute graph for each timestep **t***
1	Community detection for each graph
2	Matching detected communities for consecutive graphs
	2.1 Create joint graph (t **U** t+1)
	2.2 Community detection for joint graph
	2.3 For each detected community **v** in joint graph - Find communities in **t** and **t+1** graph contained in **v** - Calculate relative overlap for each pair - Match communities in descending order
3	Post-processing



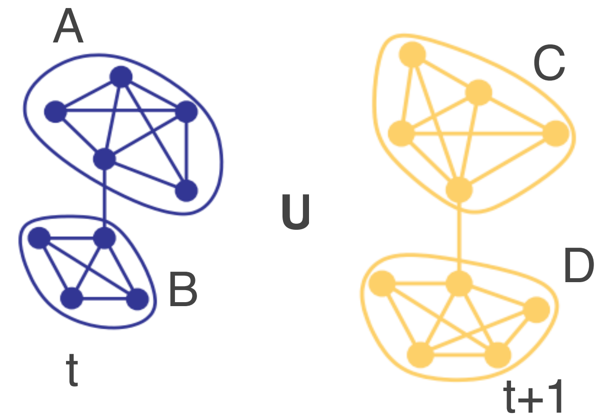
Algorithm - Community Matching

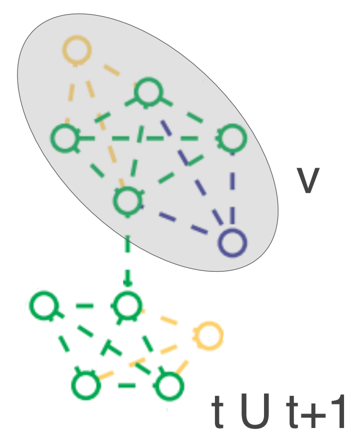
	*Compute graph for each timestep **t***
1	Community detection for each graph
2	Matching detected communities for consecutive graphs
	2.1 Create joint graph (t **U** t+1)
	2.2 Community detection for joint graph
	2.3 For each detected community **v** in joint graph - Find communities in **t** and **t+1** graph contained in **v** - Calculate relative overlap for each pair - Match communities in descending order
3	Post-processing



Algorithm - Community Matching

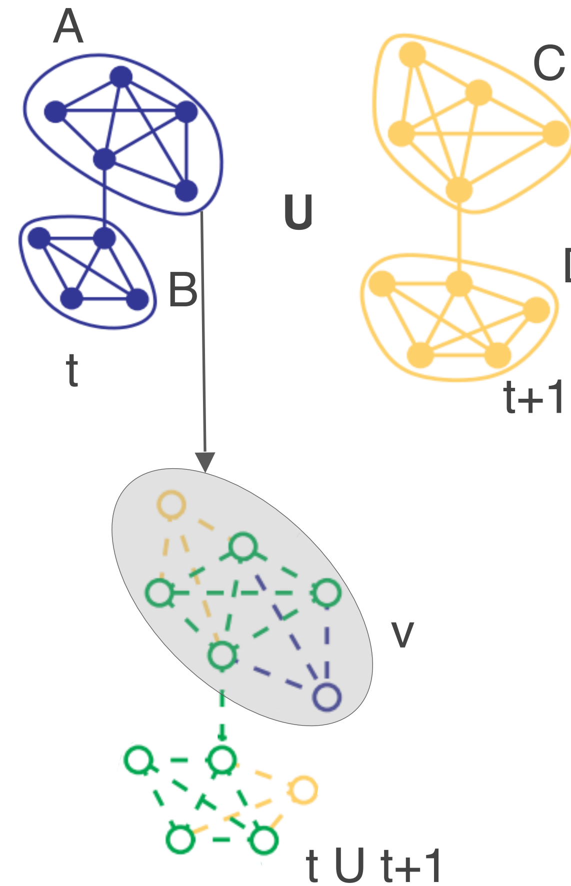
	*Compute graph for each timestep **t***
1	Community detection for each graph
2	Matching detected communities for consecutive graphs
	2.1 Create joint graph (t **U** t+1)
	2.2 Community detection for joint graph
	2.3 For **each** detected community **v** in joint graph - Find communities in **t** and **t+1** graph contained in **v** - Calculate relative overlap for each pair - Match communities in descending order
3	Post-processing





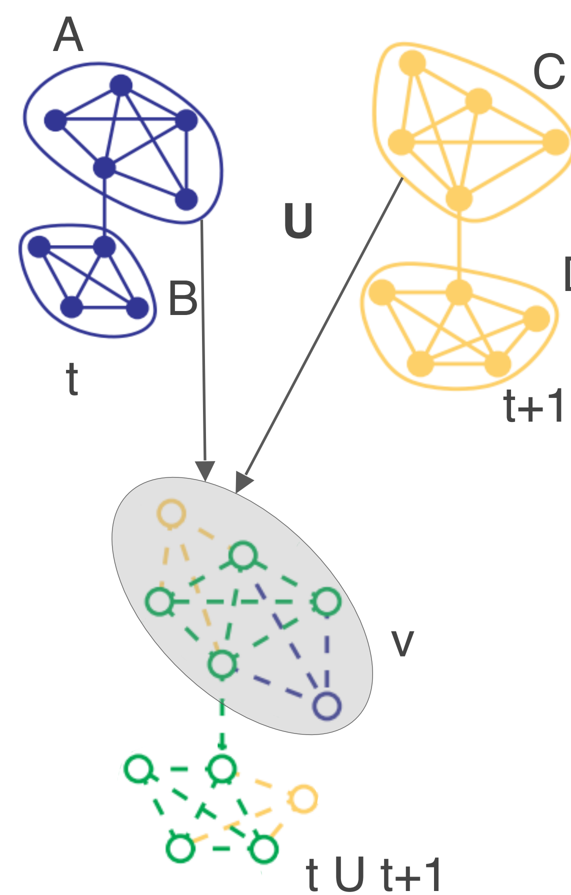
Algorithm - Community Matching

	*Compute graph for each timestep **t***
1	Community detection for each graph
2	Matching detected communities for consecutive graphs
	2.1 Create joint graph (t **U** t+1)
	2.2 Community detection for joint graph
	2.3 For **each** detected community **v** in joint graph - Find communities in **t** and **t+1** graph contained in **v** - Calculate relative overlap for each pair - Match communities in descending order
3	Post-processing



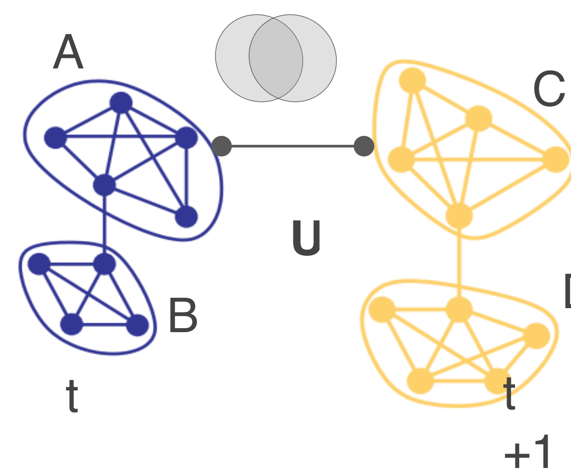
Algorithm - Community Matching

	*Compute graph for each timestep **t***
1	Community detection for each graph
2	Matching detected communities for consecutive graphs
	2.1 Create joint graph (t **U** t+1)
	2.2 Community detection for joint graph
	2.3 For **each** detected community **v** in joint graph - Find communities in **t** and **t+1** graph contained in **v** - Calculate relative overlap for each pair - Match communities in descending order
3	Post-processing



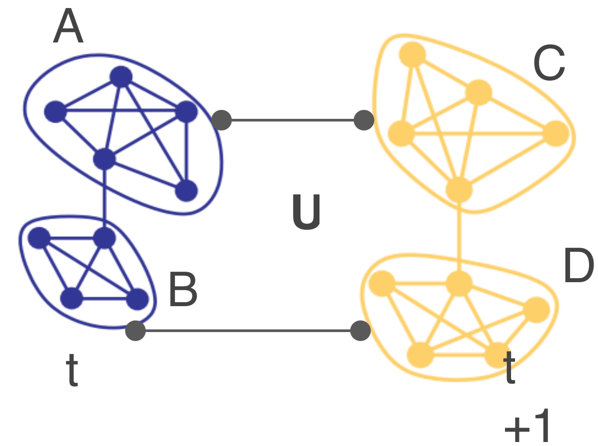
Algorithm - Community Matching

	*Compute graph for each timestep **t***
1	Community detection for each graph
2	Matching detected communities for consecutive graphs
	2.1 Create joint graph (t **U** t+1)
	2.2 Community detection for joint graph
	2.3 For **each** detected community **v** in joint graph - Find communities in **t** and **t+1** graph contained in **v** - Calculate relative overlap for each pair - Match communities in descending order
3	Post-processing



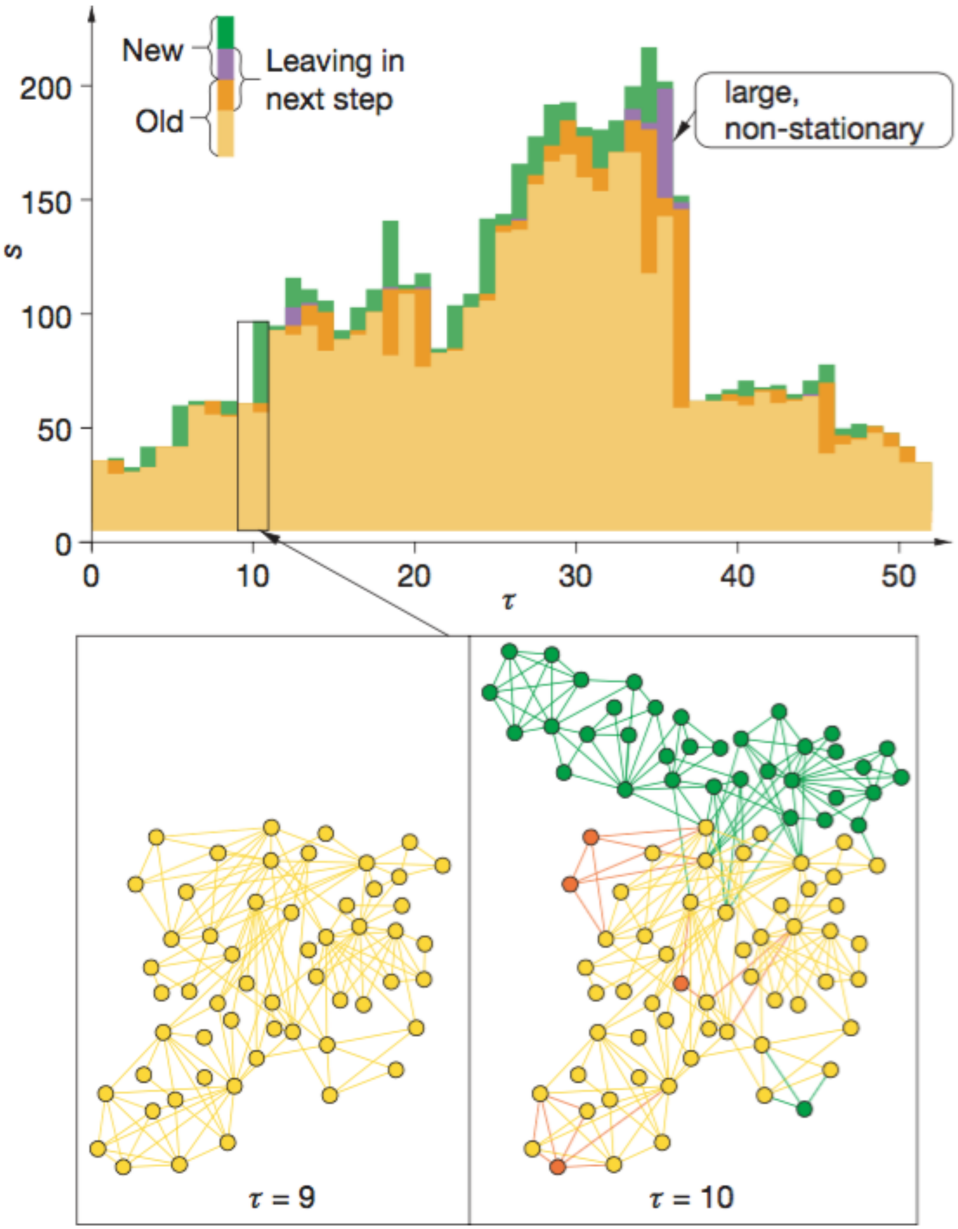
Algorithm - Community Matching

	*Compute graph for each timestep **t***
1	Community detection for each graph
2	Matching detected communities for consecutive graphs
	2.1 Create joint graph (t **U** t+1)
	2.2 Community detection for joint graph
	2.3 For **each** detected community **v** in joint graph - Find communities in **t** and **t+1** graph contained in **v** - Calculate relative overlap for each pair - Match communities in descending order
3	Post-processing



Algorithm - Community Matching

	*Compute graph for each timestep **t***
1	Community detection for each graph
2	Matching detected communities for consecutive graphs
	2.1 Create joint graph (t **U** t+1)
	2.2 Community detection for joint graph
	2.3 For **each** detected community **v** in joint graph - Find communities in **t** and **t+1** graph contained in **v** - Calculate relative overlap for each pair - Match communities in descending order
3	**Post-processing**



Evolution of communities in the co-authorship network.

The height of the columns corresponds to the actual community size.

Yellow: indicates the number of 'old' nodes (that have been present in the community at least in the previous time step as well)

Green: Newcomers

Orange: Old members abandoning the community in the next time step

Purple: New members abandoning the community in the next time step (This latter type of member joins the community for only one time step.)

Questions?