Universität Zürich[UZH]

Blockchain & Distributed Ledger Technologies

UZH Blockchain Center

Communities and Community Detection Algorithms

Lecture 4

Claudio J. Tessone

Blockchain & Distributed Ledger Technologies
UZH Blockchain Center

Outlook

1. Introduction
2. Modularity
3. Community detection
4. Algorithms comparison

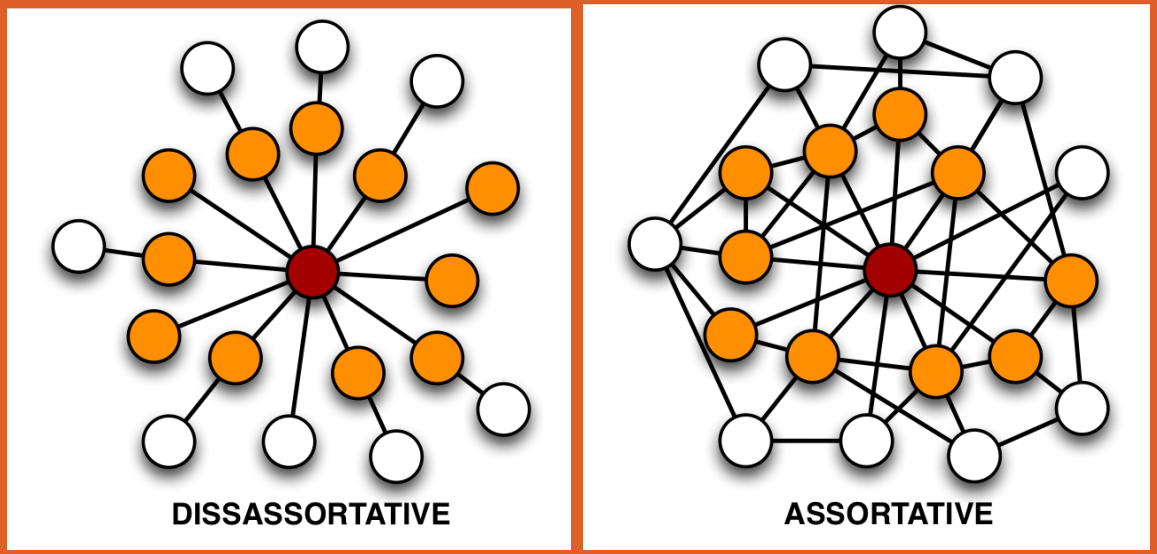
Lecture Objectives

1. Get acquainted with the concept of network community
2. Learn how to measure the importance of communities within a network?
3. Which methods allow to detect communities?

Empirical analysis: Degree

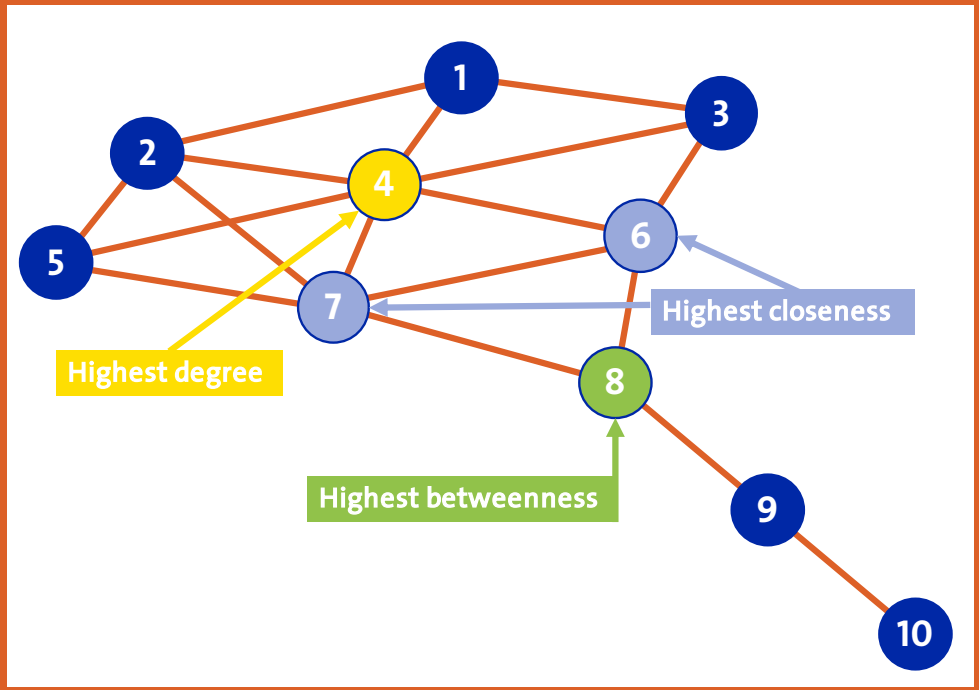
Node degree gives only local information. Degree distribution is a global description

Empirical analysis: Higher order properties



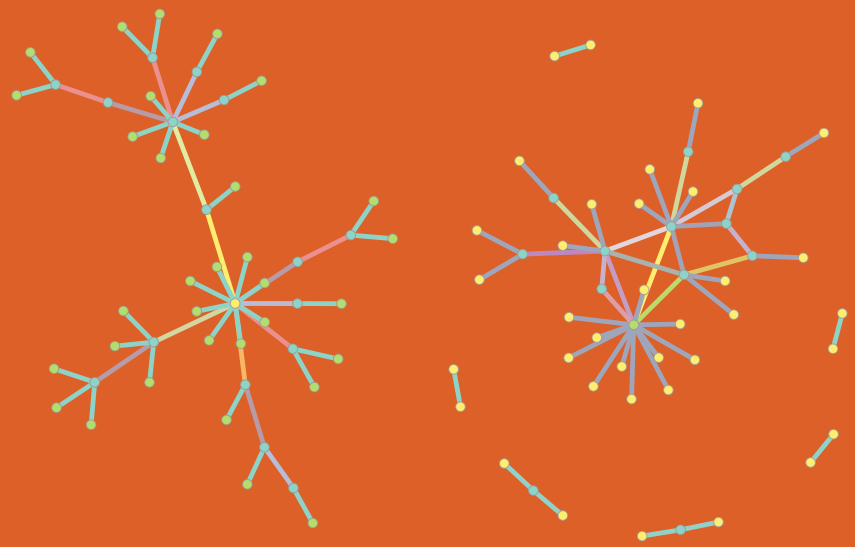
Assortativity, clustering tell us how nodes are interconnected

Empirical analysis: Node importance



Centrality can be a local or global concept, but encodes different (multiple) information about what makes important a node

Randomisation



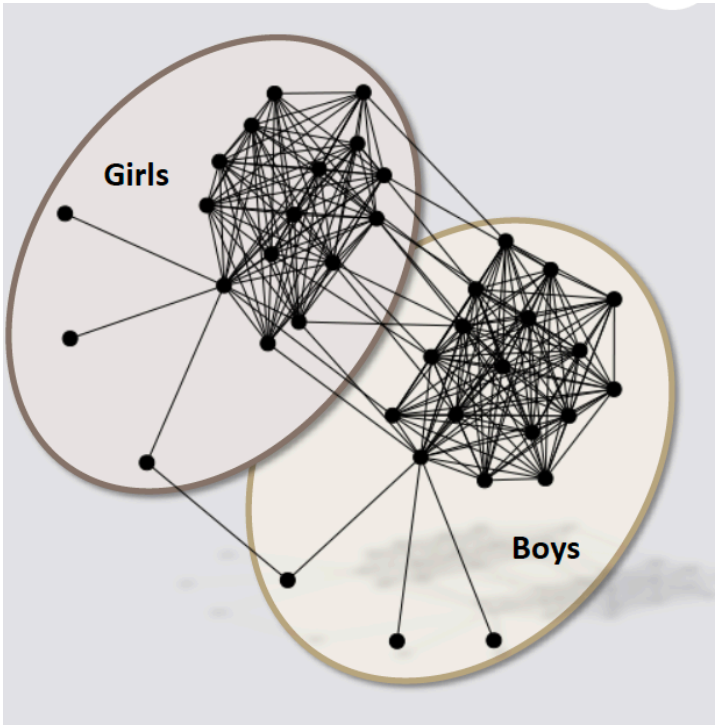
Randomising a network allows us to understand if the properties we measure are the result of simple statistical properties

So far

1. Nodes are all treated in the same way
2. There is no label / differentiating factor that is inherent to them

Introduction

But nodes represent entities...



In this network, nodes are children, edges are friendships **and node sets represents their gender**

A **community** is a group of system constituents who share common properties and/or play similar roles within the system

> *Nodes within the same community tend to be connected with each other much more frequently within the community than with nodes outside*

We can represent these communities e.g. by drawing the nodes with different colours

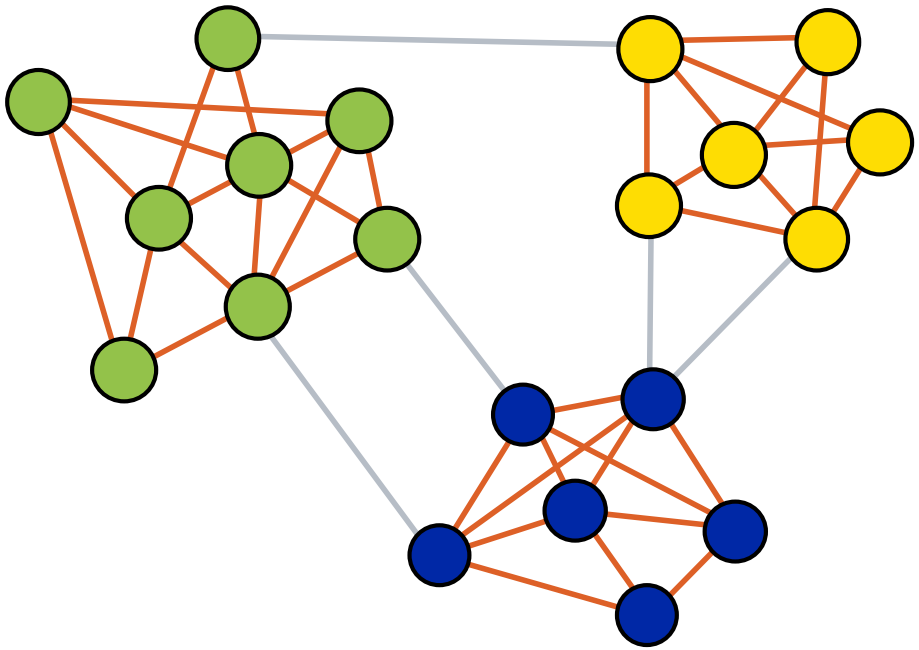
A **community** is a group of system constituents who share common properties and/or play similar roles within the system

> *Nodes within the same community tend to be connected with each other much more frequently within the community than with nodes outside*

We can represent these communities e.g. by drawing the nodes with different colours

Communities or clusters

From a structural point of view communities are sets of tightly connected nodes

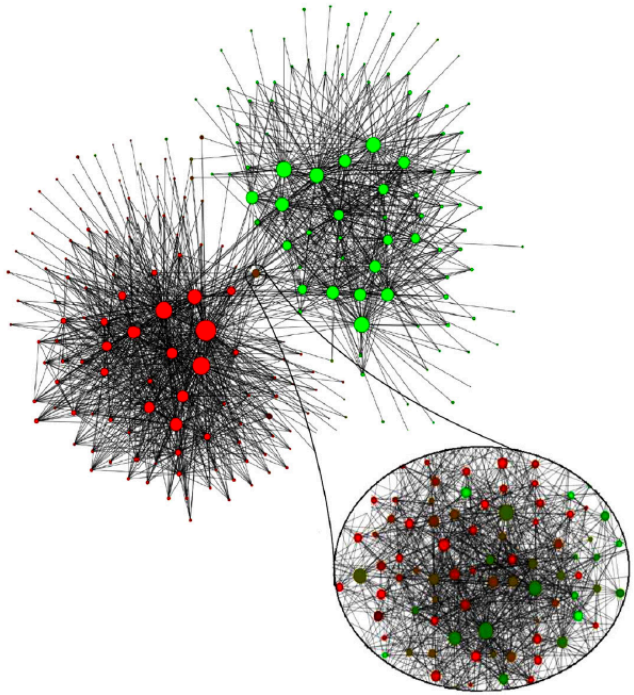


In real world

Real world networks often have the feature that the nodes of the network can be grouped into sets of densely connected nodes, which show the community structure

- Collaboration networks (scientific, R&D)
- Social networks
- Co-purchases
- Protein interaction networks

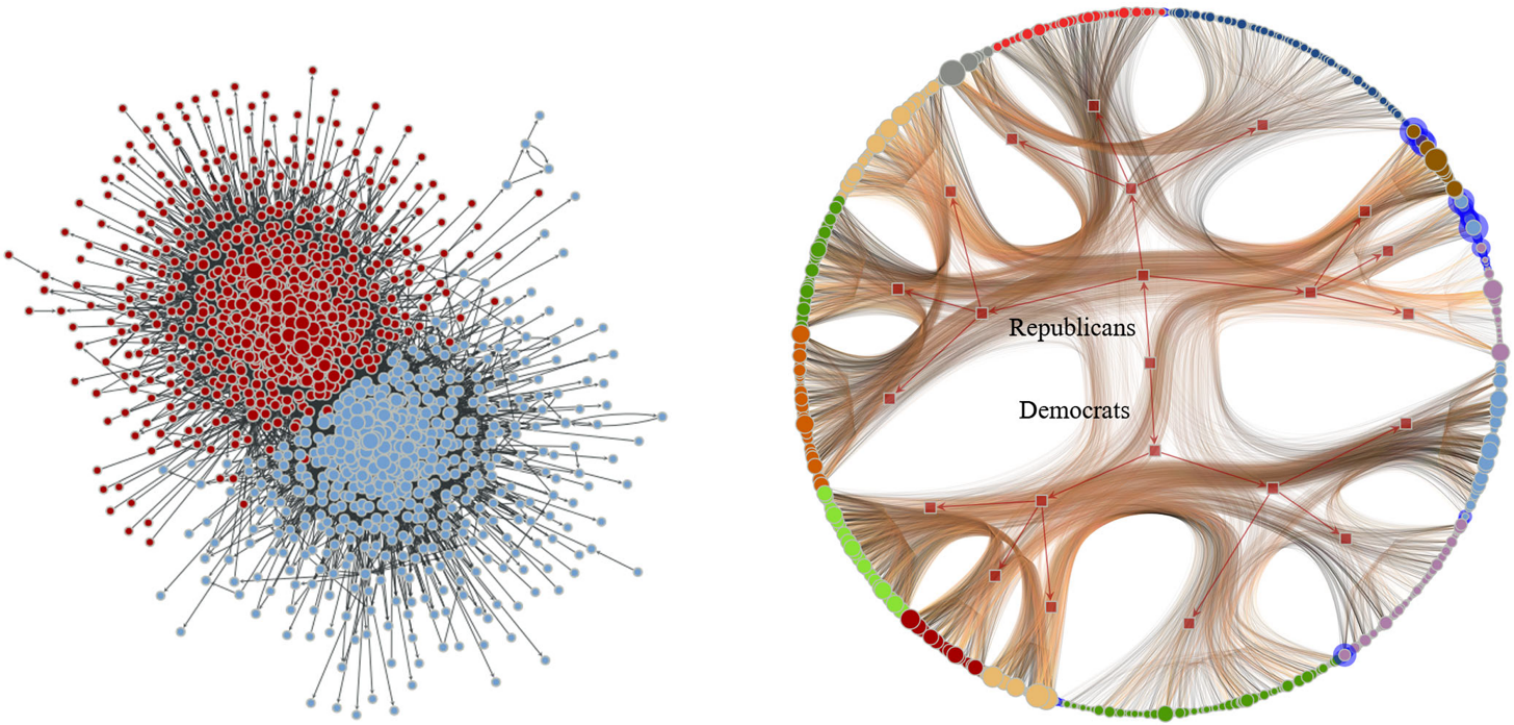
Social networks



Belgian phone communication network: Nodes are persons, edges exist if phone communication existed between them. Colour represent mother tongue

Two clearly defined groups of people speaking the different languages

Political discussion

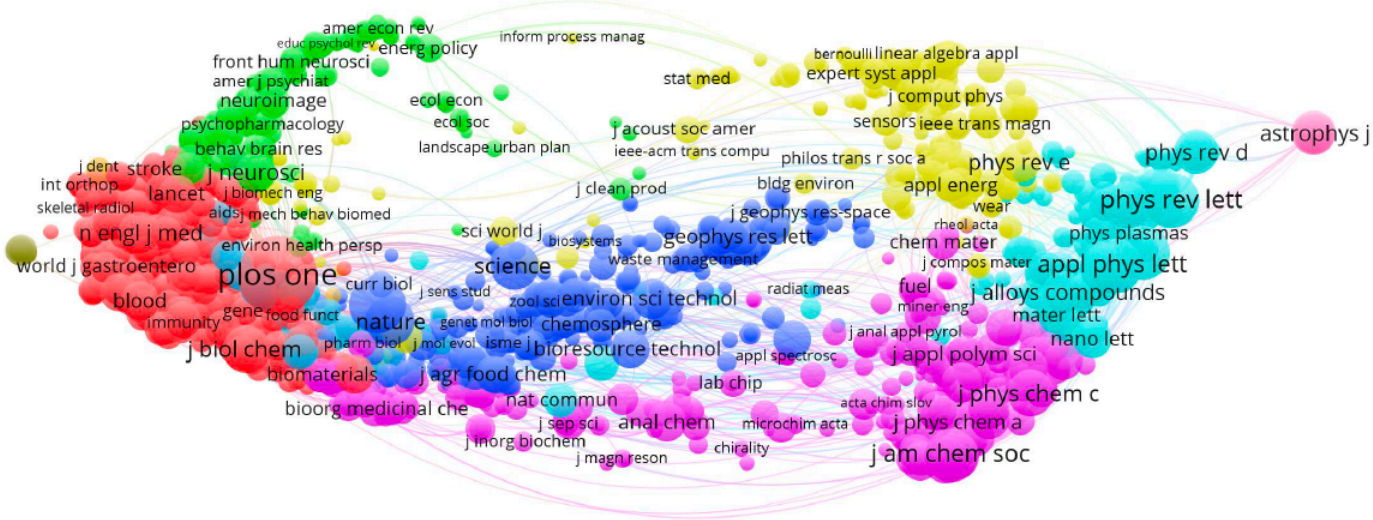


Network of political blogs during the 2004 presidential election in the USA. Nodes are authors, links exist if they cite each other. Colour represents supported party

Two clearly distinguishable communities: Democrats and Republicans

[Tiago P. Peixoto, *Hierarchical block structures and high-resolution model selection in large networks* (2014)]

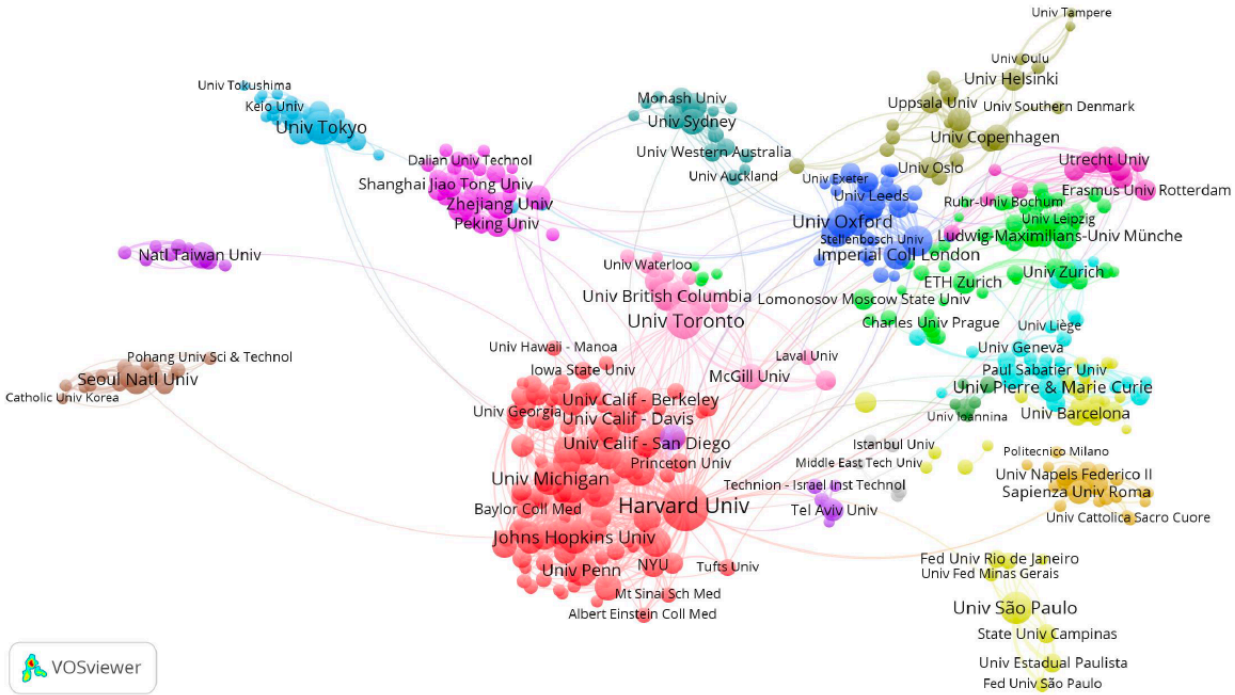
Map of science



Journal citation network: Nodes are journals, edges mean citations between them,

Colours represent knowledge area

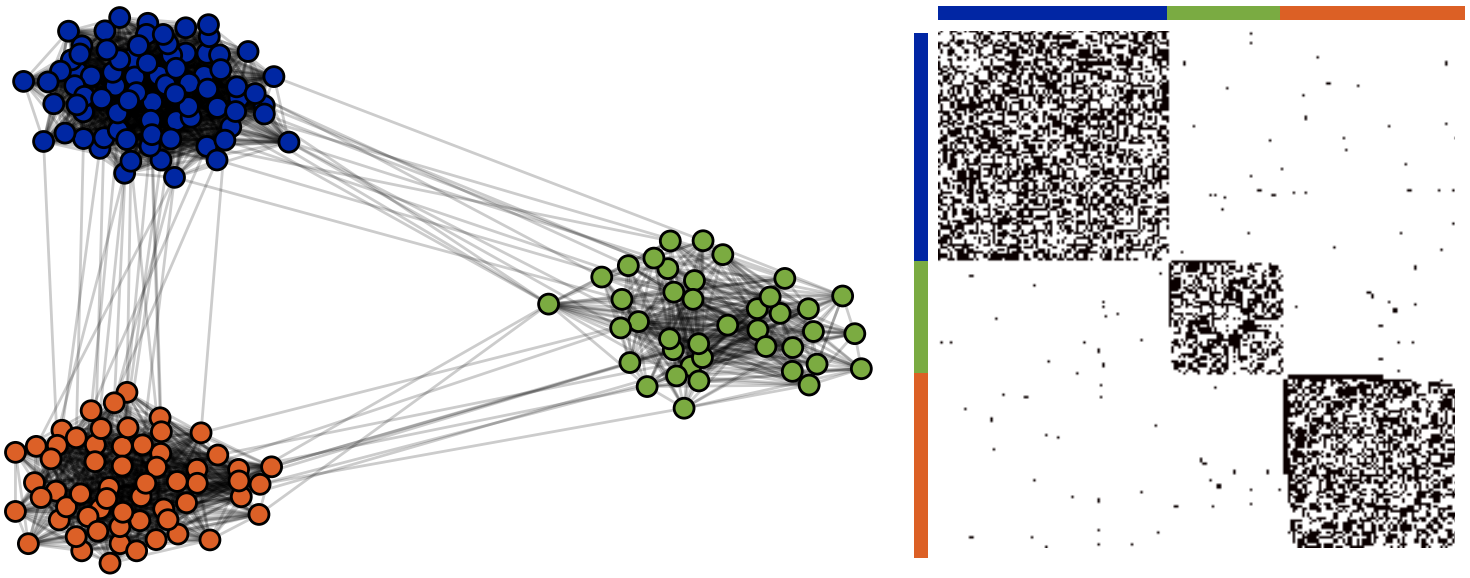
Map of science



University collaboration network: Nodes are Universities, edges mean collaboration between them, Colours represent geographical areas

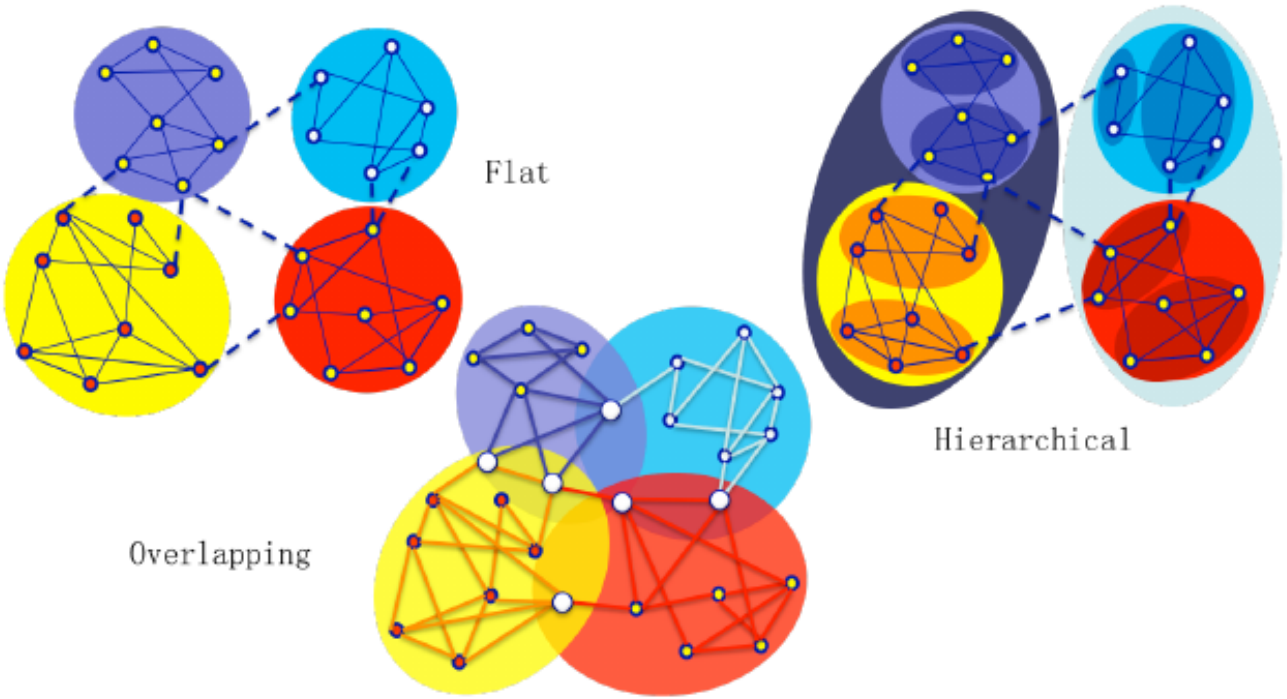
Why do nodes cluster into communities? Common features (e.g. demographics), Segregation / Polarisation

Community recap



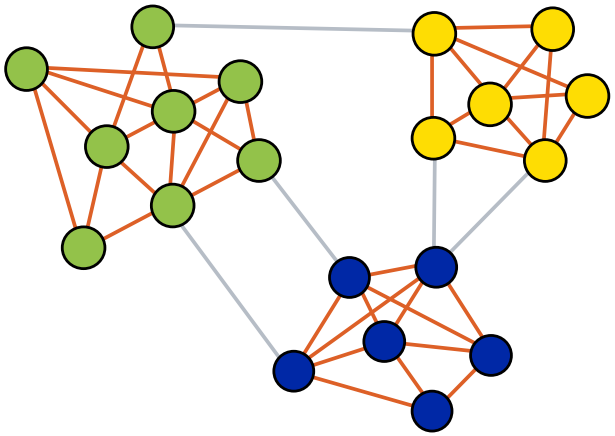
Communities look like densely connected blocks in the adjacency matrix

Community recap



In this lecture we focus on flat, **non-hierarchical** communities.

Main properties of communities



- *High cohesion: communities have many internal links, so their nodes stick together*
- *High separation: communities are connected to each other by few links*

How to quantify them?

Modularity

Basic definitions: variables

Internal degree of a node: number of neighbours of the node in its community

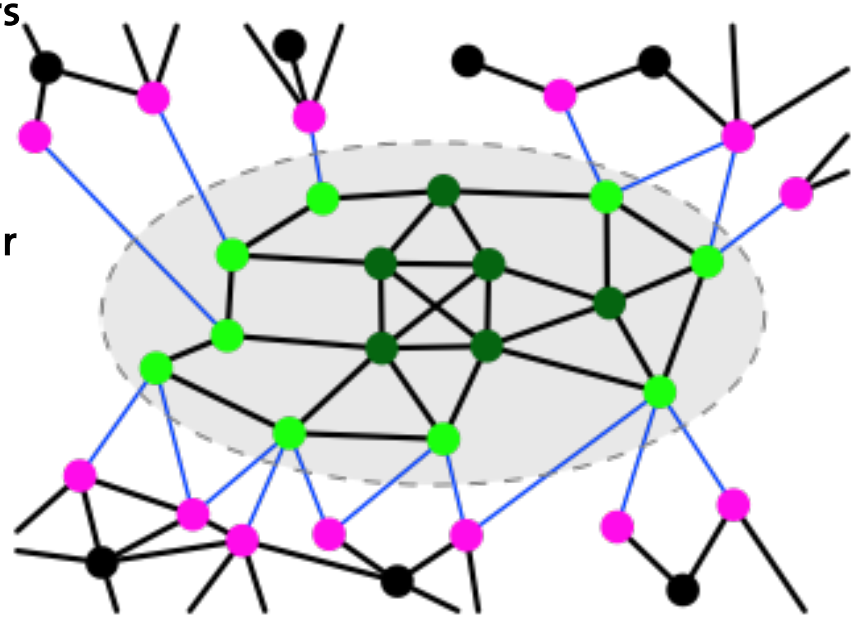
External degree of a node: number of neighbors of the node outside of its community

Community degree: sum of the degrees of the nodes in the community

Internal link density: ratio between the number of links L_C inside a community C and the maximal possible number of links that can lie inside C:

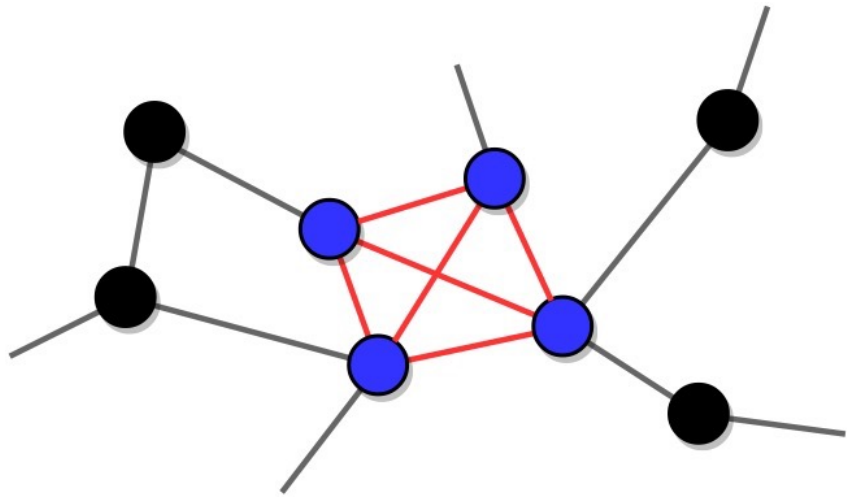
$$\delta_C^{int} = \frac{L_C}{L_c^{max}} = \frac{L_C}{\binom{N_C}{2}} = \frac{2L_C}{N_C(N_C-1)}$$

where N_C is the number of nodes in C



Community definitions based on cohesion

- Principle: focus on the cluster's properties, disregarding the rest of the network
- Example: clique — all internal links are there (maximal cohesion)
- Problem: nodes are connected to all others in the cluster, whereas in real communities they have different roles, which is reflected in heterogenous linking patterns



Community definitions based on cohesion versus separation

- Principle: definition tends to achieve high cohesion and high separation
- Popular idea: the number of internal links exceeds the number of external links

Strong community

Subnetwork such that the internal degree of each node is greater than its external degree

Weak community

Subnetwork such that the sum of internal degrees of its nodes is greater than the sum of their external degree

Community definitions based on cohesion versus separation

A strong community is also a weak community: if the inequality between internal and external degree holds for each node, then it must hold for the sum over all nodes

A weak community is not a strong community, in general: if the inequality between internal and external degree holds for the sum, it may be violated for one or more nodes

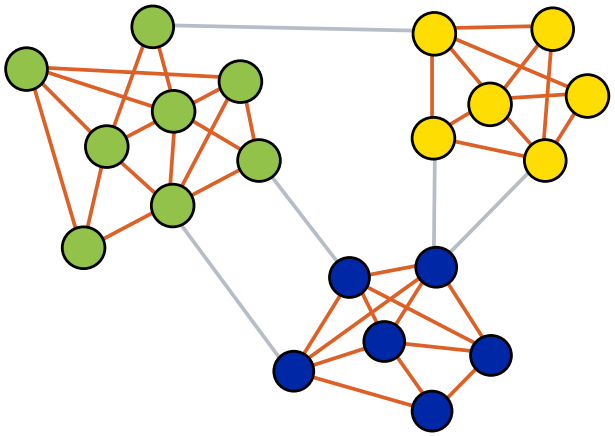
In the definition of strong and weak community one compares a subnetwork with the rest of the network. It makes more sense to compare subnetworks to each other!

Modularity: idea

- ***Principle:*** evaluating communities with respect to a random baseline
- ***Baseline:*** randomised version of the original network, but preserving its degree sequence

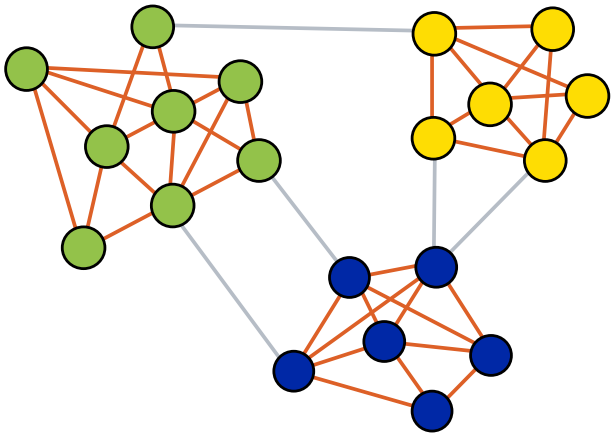
For each community of a partition, modularity computes the difference between the number of internal links in the community and the expected value of this number in the set of randomised networks

Modularity: intuition



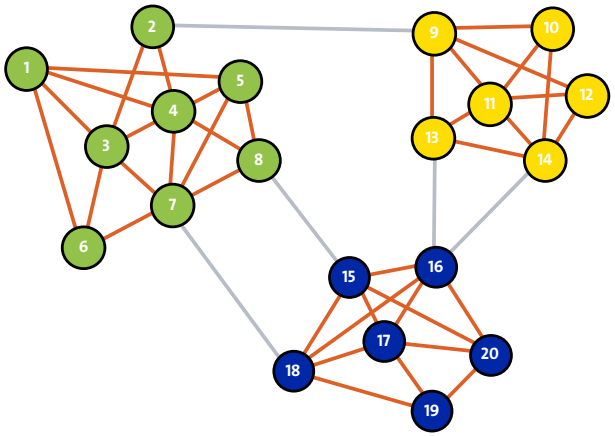
If the network is random (c.f. Lecture 5), the modularity of any partition should be low; the number of internal links of any cluster of the partition should be close to the expected value in the randomised networks

Modularity: intuition



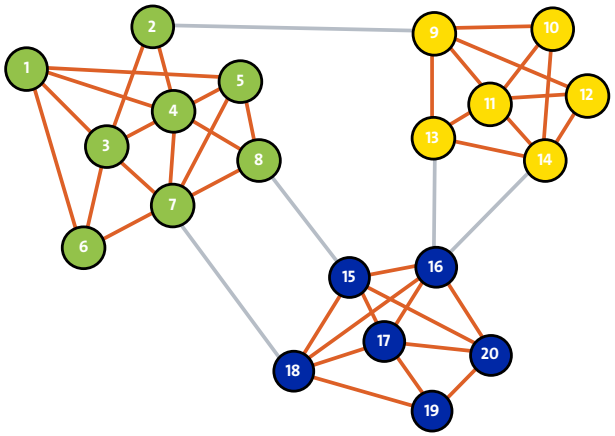
If the number of links within the clusters is much larger than its expected random value, it is unlikely for such a concentration of internal links to be the result of a random process, and modularity can reach high values

Modularity score



- Given a network with N nodes, and L edges
- We start from a *partition of the nodes*, each node is assigned to a community C
- In the example $C \in [1, 2, 3]$

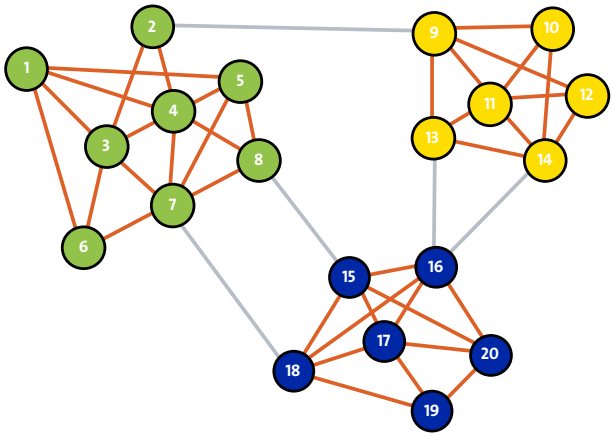
Modularity score



$k_C = \sum_{i \in C} k_i$ is the total degree of community C

$L_C = \sum_{i,j \in C} a_{i,j}$ number of internal links in community C

Modularity score

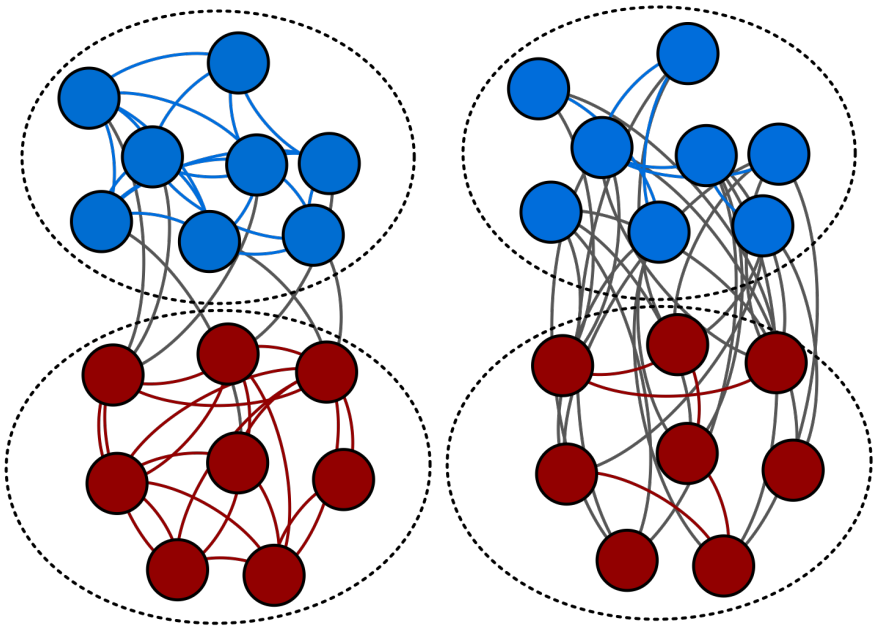


Then, for a random network, the expected number of internal links in community is

$$\frac{k_C^2}{4L}$$

Modularity

- Let's explain the origin of $k_c^2/4L$
- Random links are formed by matching pairs of stubs (half-links) chosen at random
- The total number of stubs attached to C is k_C
- The probability to select one of those stubs at random is $k_c/2L$ because 2L is the total number of stubs of the network (each link yields two stubs)



Original network Randomised network

Modularity

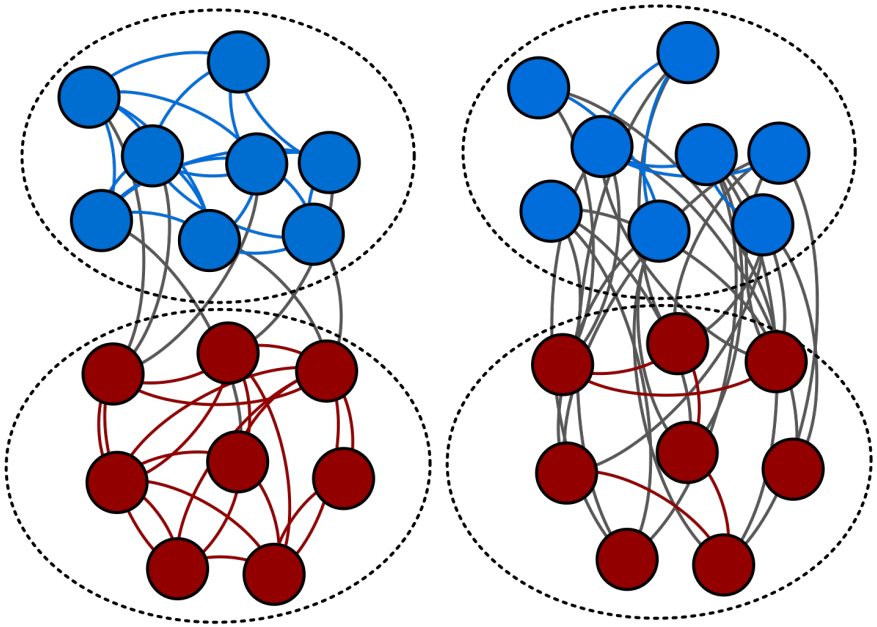
- For a random link to connect two nodes in the same cluster C, two stubs must be selected from C

- The probability to pick two stubs from C at random is (roughly) the product of the probabilities of selecting each one:

$$p_c = \left(\frac{k_C}{2L}\right)\left(\frac{k_C}{2L}\right) = \frac{k_C^2}{4L^2}$$

- Since there are L links in the network, and each has a probability p_C to end up within C, the expected number of internal links in C is

$$p_C\, L = \frac{k_C^2}{4L^2} L = \frac{k_C^2}{4L}$$



Original network Randomised network

Modularity

$$Q = \frac{1}{L} \sum_{C} \left(L_C - \frac{k_C^2}{4L} \right)$$

L	*Number of links in the network*
L_C	*Total degree of community C*
k_c	*Number of internal links in community C*
$k_c^2/4L$	*Expected number of internal links in community C*

Modularity matrix

$$B_{ij} = A_ij - \frac{k_i\, k_j}{2L}$$

It gives the difference between the adjacency matrix and the expected number of links between nodes i and j

Modularity: features

- *Q < 1 for every partition of any network*
- *Q = 0 for the partition in which the whole graph is one community*
- *Q can be negative (e.g., partition in N groups of one node each)*
- *For most networks, Q has a non-trivial maximum between 0 and 1*

```
# returns the modularity of the input partition
modularity = nx.community.quality.modularity(G,partition)
```

Modularity: extension for directed network

$$Q_d = \frac{1}{L} \sum_C \left(L_C - \frac{k_C^{in} k_C^{out}}{L} \right)$$

L	*Number of links in the network*
L_C	*Total degree of community C*
k_C^{in}	*Total in-degree in community C*
k_C^{out}	*Total out-degree in community C*

Modularity: extensions

Weighted networks

$$Q_w = \frac{1}{W} \sum_C \left(W_C - \frac{s_C^2}{4W} \right)$$

- *W = total weight of network links*
- *W_C = total weight of internal links in community C*
- *s_C = total strength of nodes in community C, i.e., sum of the strengths of the nodes in C*

Modularity: extensions

Directed and weighted networks

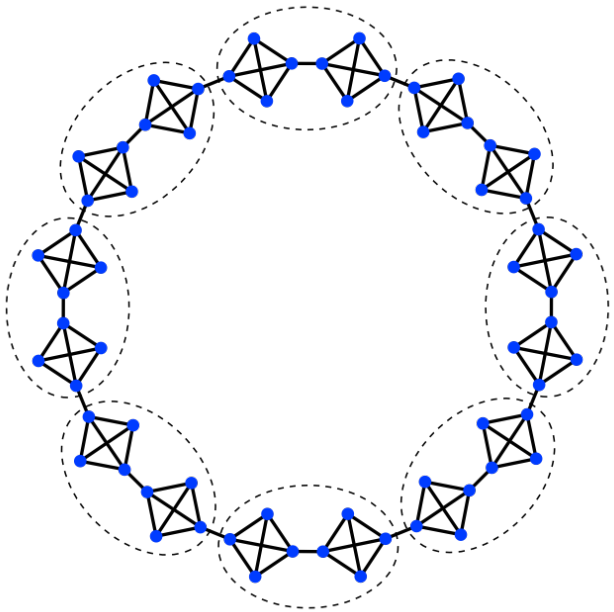
$$Q_{dw} = \frac{1}{W} \sum_C \left(W_C - \frac{s_C^{in} s_C^{out}}{W} \right)$$

- *W = total weight of network links*
- *W_C = total weight of internal links in community C*
- *s^{in}_C = total in-strength of nodes in community C*
- *s^{out}_C = total out-strength of nodes in community C*

Modularity optimisation: limits

- The maximum modularity tends to be larger on larger networks, so the measure cannot be used to compare the quality of partitions across networks
- The maximum modularity of random networks without group structure (e.g., Erdős–Rényi) can attain fairly large values
- The maximum modularity does not necessarily correspond to the best partition. Communities smaller than a certain size may not be detected (resolution limit)

Modularity: resolution limit



The natural partition is the one where the communities are the cliques but modularity is larger for the partition where pairs of cliques are merged

- Possible solution: tuning the resolution of the method by inserting a parameter in the modularity formula (multiresolution modularity optimization)
- Two problems:
  - Computationally intensive: modularity has to be optimized for multiple choices of the resolution parameter
  - A criterion is needed to decide which value of the resolution parameter is most suitable for a given network

This problem stems from the inherent difficulty in translating a qualitative notion (social community) into a mathematical definition in a complex system like a network

Community detection

Network Partitions

- The number of partitions of *n* objects is the Bell number

$$B_{n+1} = \sum_{k=0}^{n} \binom{n}{k} B_k$$

- The Bell number grows faster than exponentially with *N*

n	B_n
1	1
2	2
3	5
4	15
5	52
6	203
7	877
8	4140
9	21147
10	115975
11	678570
12	4213597
13	27644437
14	190899322
15	1382958545

It does not make sense to look for community structures by systematic exploration in the whole space of partitions! A constrained, smart exploration of the partition space must be performed.

Community detection refers to discovering groups in a network where individuals' group memberships are not explicitly given

It aims at unfolding the logical communities by only using the structural properties of the network

Community detection

- **Main assumption**: Individuals belonging to one community are connected:
  - more densely with those within the community
  - less densely with those outside the community
- In this lecture we are focusing on algorithms detecting **flat** communities

Community detection algorithms

- Modularity-based algorithms
  - Leading eigenvector
  - Fast greedy
  - Multilevel
- Random-walk based algorithm
  - Walk trap
- Other algorithms
  - Edge betweenness
  - Label propagation

Algorithms overview

- **Stable algorithms**: produce the same results if they are run on the same graph multiple times

	Algorithm	Stable	Fast
Modularity based	Leading eigenvector	No	No
	Fast greedy	Yes	Yes
	Multilevel	Yes	Yes
Random walk	Walktrap	Yes	No
Other	Edge betweenness	Yes	No
	Label propagation	No	Yes

Label propagation

Idea

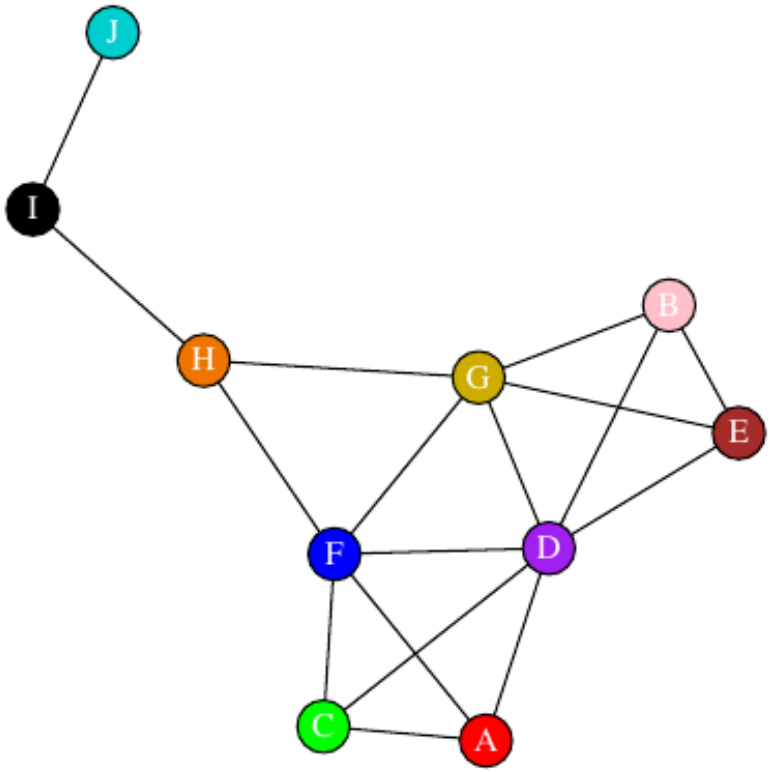
Assign each node in the network to the community to which belongs the majority of its neighbours

Label Propagation: Algorithm

1. Initialise a distinct community for each node;
2. Arrange nodes in random order (without replacement)
3. Assign each node the label of the majority of its neighbours
4. Move in the pre-assigned random order
5. Break ties randomly
6. Stop if every node already has the label of majority of its neighbours

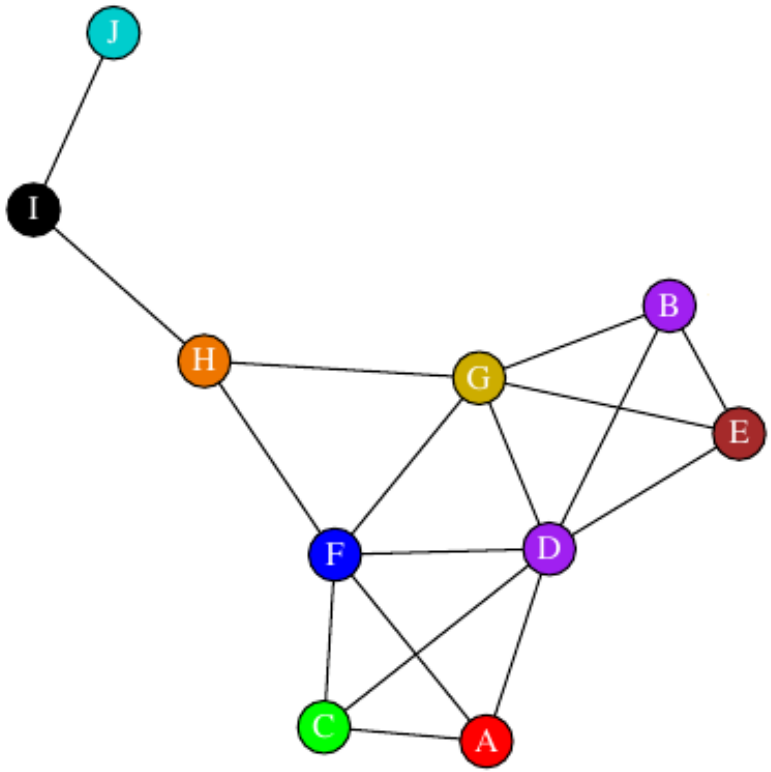
Label Propagation example

Initialise with singleton communities



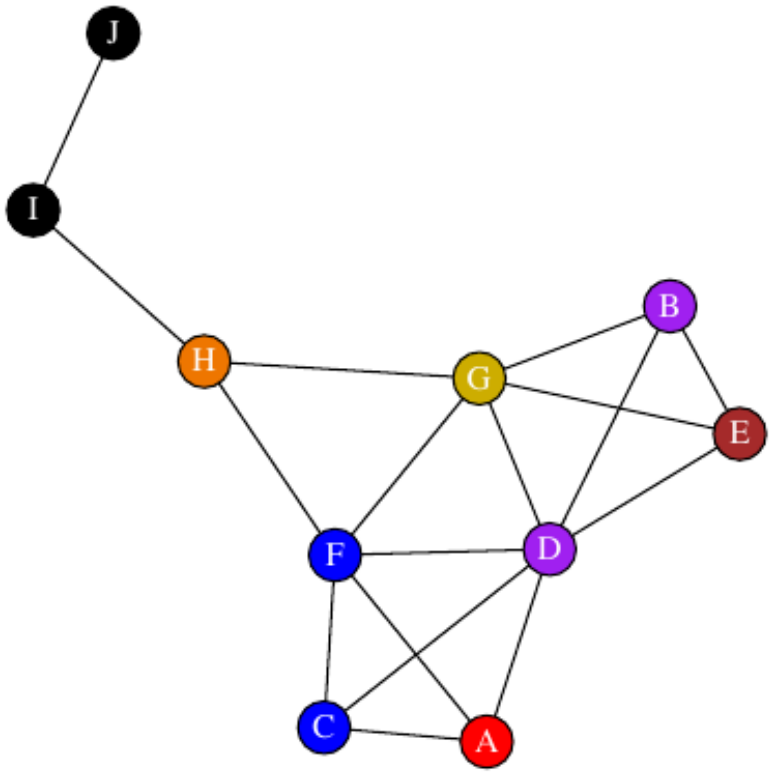
Label Propagation example

For a randomly chosen node (B), assign label of its neighbours majority (as D, ties broken randomly)



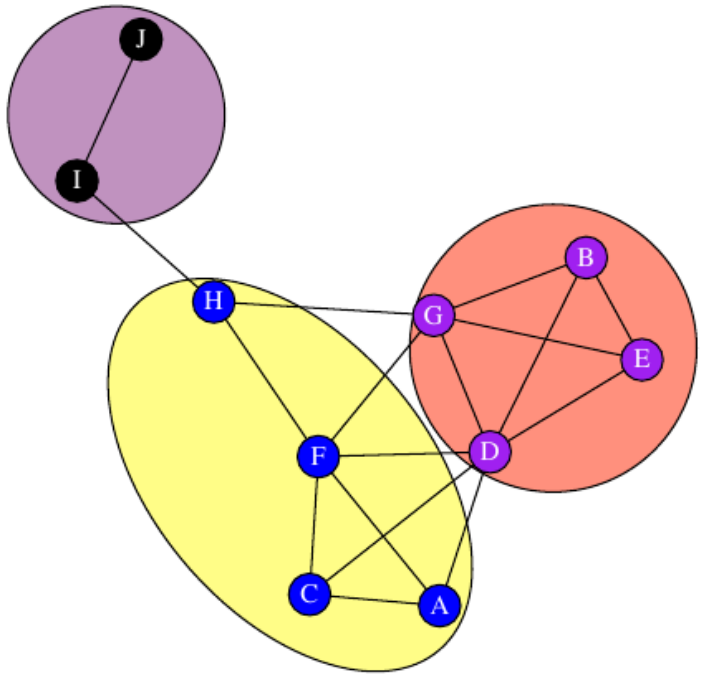
Label Propagation example

For a randomly chosen node (J), assign label of its neighbours majority (as I, its only neighbour)



Label Propagation example: result

Proceed until all nodes are in the same communities as the majority of their neighbours.



Label Propagation algorithm properties

- Results depend to some extent in the random sequence order
- Strategy for tie-breaking also matters

Edge betweenness

Idea

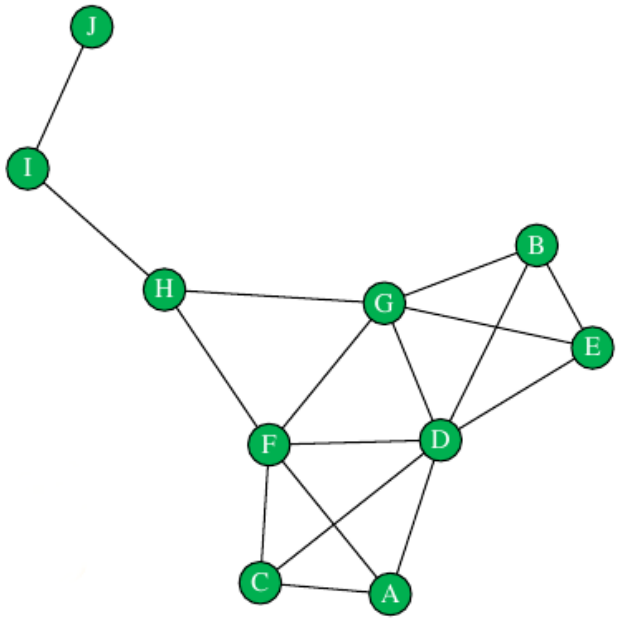
detect communities by progressively removing edges with the highest edge betweenness centrality from the original network

Edge betweenness

1. Identify edge with highest edge betweenness
2. Delete it edge
3. Re-calculate edge-betweenness for every edge of the updated network
4. Repeat until no edges left
5. Choose the community structure with the highest modularity score among all possible structures

Edge betweenness algorithm

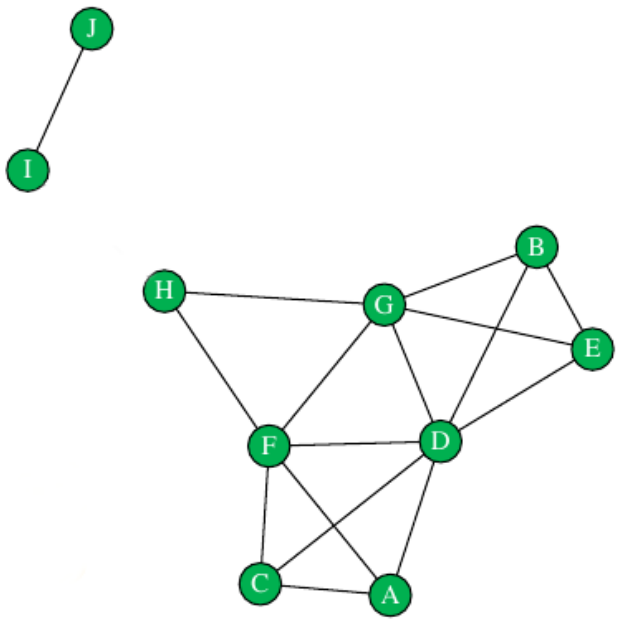
Calculate betweenness score of all edges



Edge	Value
D - A	3.5
C - A	1
F - A	4.5
C - D	3.5
F - D	3.5
B - D	3.5
E - D	3.5
G - D	3.5
F - C	4.5
G - F	3
H - F	10.5
E - B	1
G - B	4.5
G - E	4.5
H - G	10.5
I - H	16
J - I	9

Edge betweenness algorithm

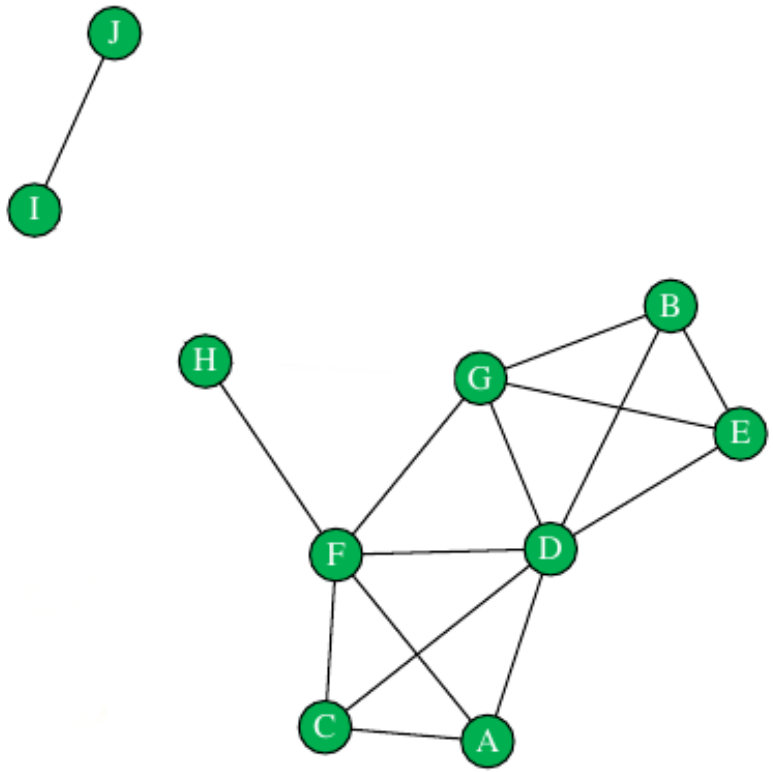
Delete the edge with highest betweenness score & recalculate betweenness for all edges



Edge	Value
D - A	3.5
C - A	1
F - A	2.5
C - D	3.5
F - D	2.5
B - D	3.5
E - D	3.5
G - D	2.5
F - C	2.5
G - F	3
H - F	3.5
E - B	1
G - B	2.5
G - E	2.5
H - G	3.5
J - I	1

Edge betweenness algorithm

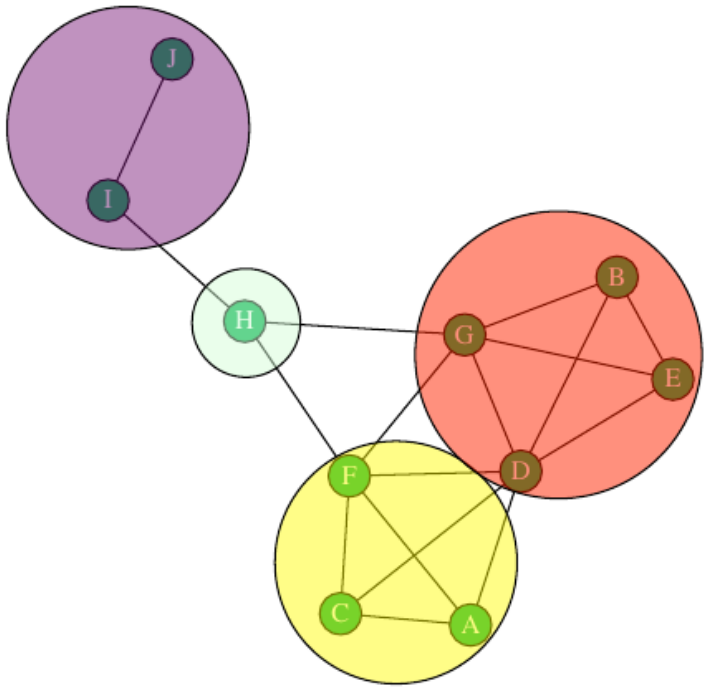
Break ties randomly; Proceed until all edges are removed.



Edge	Value
D - A	3.5
C - A	1
F - A	2.5
C - D	3.5
F - D	4
B - D	4
E - D	4
G - D	2
F - C	2.5
G - F	5
H - F	7
E - B	1
G - B	2
G - E	2
J - I	1

Edge betweenness algorithm: result

Keep track of the modularity score. Stop, if it does not improve.



Edge betweenness algorithm: properties

- Slow algorithm; Complexity is $O(m^2 n)$
- Makes a lot of redundant calculations
- No intrinsic stopping criteria, only external (modularity)

Leading Eigenvector Algorithm

Idea

Bisect the nodes into partitions while modularity increases. Nodes that belong to the same community will contribute in the same way to the leading eigenvalue of the Modularity matrix

Leading Eigenvector Algorithm

1. Calculate eigenvector corresponding to the largest eigenvalue;
2. Split vertices into 2 communities based on the sign of the corresponding element of the eigenvector;
3. Calculate modularity improvement;
4. Repeat for both parts while modularity score improves;
5. Stop once modularity improvement is negative

Leading Eigenvector example

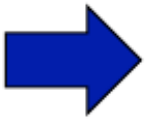
Modularity matrix:

	A	B	C	D	E	F	G	H	I	J
A	-0.265	-0.265	0.735	0.471	-0.265	0.559	-0.441	-0.265	-0.176	-0.088
B	-0.265	-0.265	-0.265	0.471	0.735	-0.441	0.559	-0.265	-0.176	-0.088
C	0.735	-0.265	-0.265	0.471	-0.265	0.559	-0.441	-0.265	-0.176	-0.088
D	0.471	0.471	0.471	-1.059	0.471	0.118	0.118	-0.529	-0.353	-0.176
E	-0.265	0.735	-0.265	0.471	-0.265	-0.441	0.559	-0.265	-0.176	-0.088
F	0.559	-0.441	0.559	0.118	-0.441	-0.735	0.265	0.559	-0.294	-0.147
G	-0.441	0.559	-0.441	0.118	0.559	0.265	-0.735	0.559	-0.294	-0.147
H	-0.265	-0.265	-0.265	-0.529	-0.265	0.559	0.559	-0.265	0.824	-0.088
I	-0.176	-0.176	-0.176	-0.353	-0.176	-0.294	-0.294	0.824	-0.118	0.941
J	-0.088	-0.088	-0.088	-0.176	-0.088	-0.147	-0.147	-0.088	0.941	-0.029

Leading Eigenvector example

Eigenvalues
1.732
1.681
0.588
0.001
-0.791
-1
-1
-1.610
-1.732
-1.870

Leading eigenvector
4.440369e-01
-4.440369e-01
4.440369e-01
5.162537e-15
-4.440369e-01
3.250576e-01
-3.250576e-01
-4.773959e-15
-7.244205e-15
-5.176415e-15

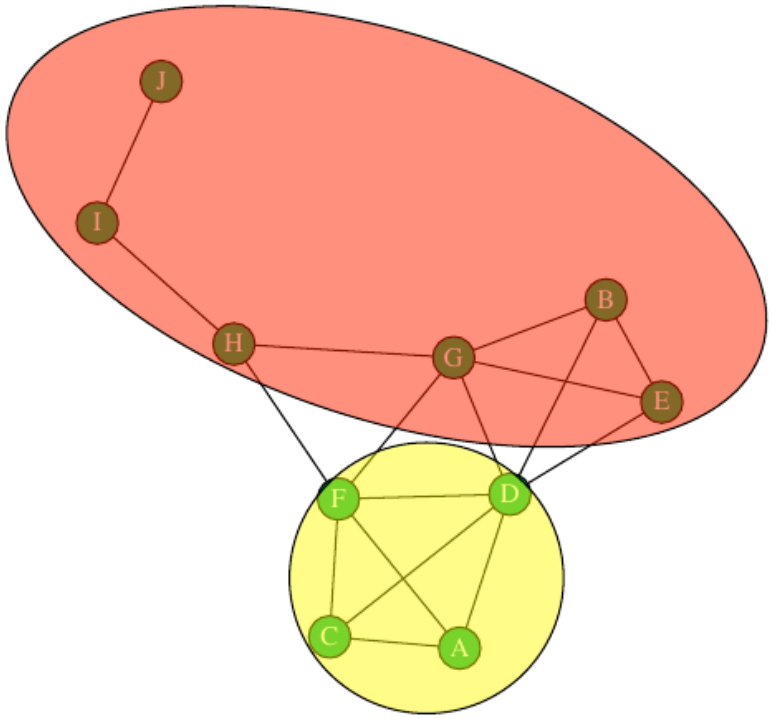


Signs of elements
+
-
+
+
-
+
-
-
-
-

Group	Node
1	A
2	B
1	C
1	D
2	E
1	F
2	G
2	H
2	I
2	J

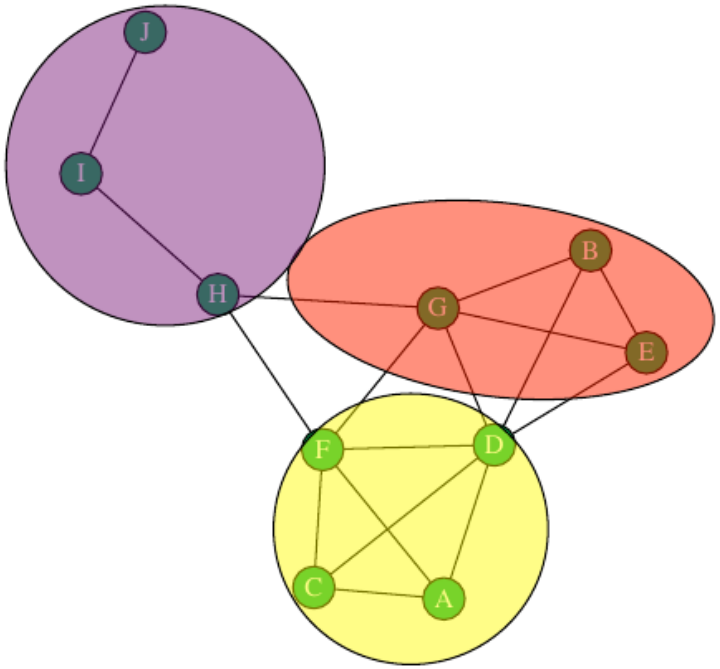
Leading Eigenvector result

Resulting communities. Modularity of this partition is 0.21.



Leading Eigenvector result

In the second iteration the modularity of the partition is



increased to 0.26.
Further iterations did not increase the modularity.

Leading Eigenvector algorithm: properties

- Depends heavily on the eigenvector calculation routine, which is often unstuble
- Might split tightly connected groups

A random walk in a network

1. Start at a specified vertex
2. Take d step random walk:
   - **2.1** Choose uniformly at random between the edges attached to the node
   - **2.2** Follow this edge to the next node

Walk trap algorithm

Idea

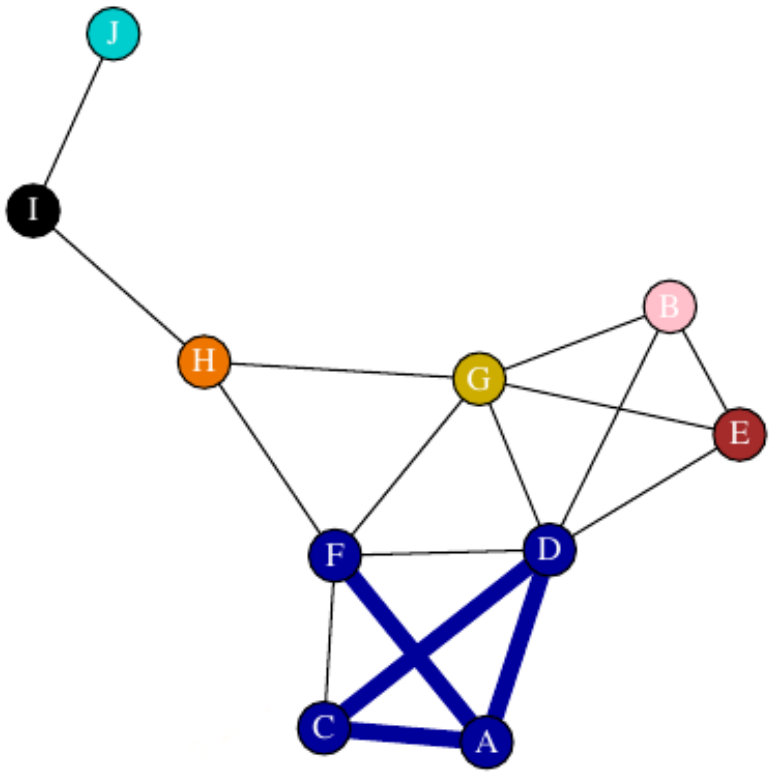
short-distance random walk tends to stay in the same community

Walk trap algorithm

- Algorithm: runs a number of short random walks
- Merge communities in a bottom-up manner
- Stopping criterion: *modularity score decreases*

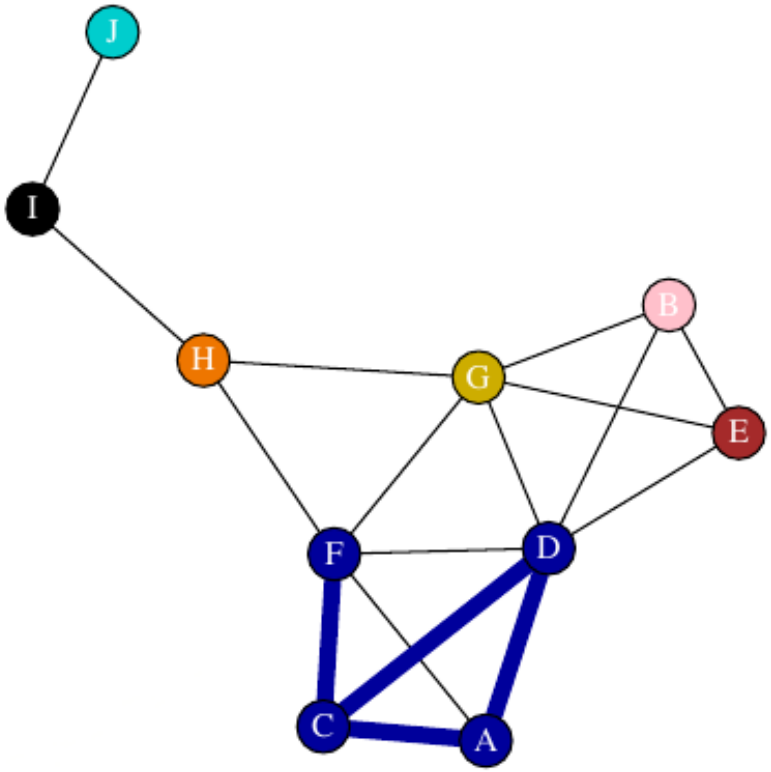
Walk trap algorithm

One random walk of length 4: A - C - D - A - F

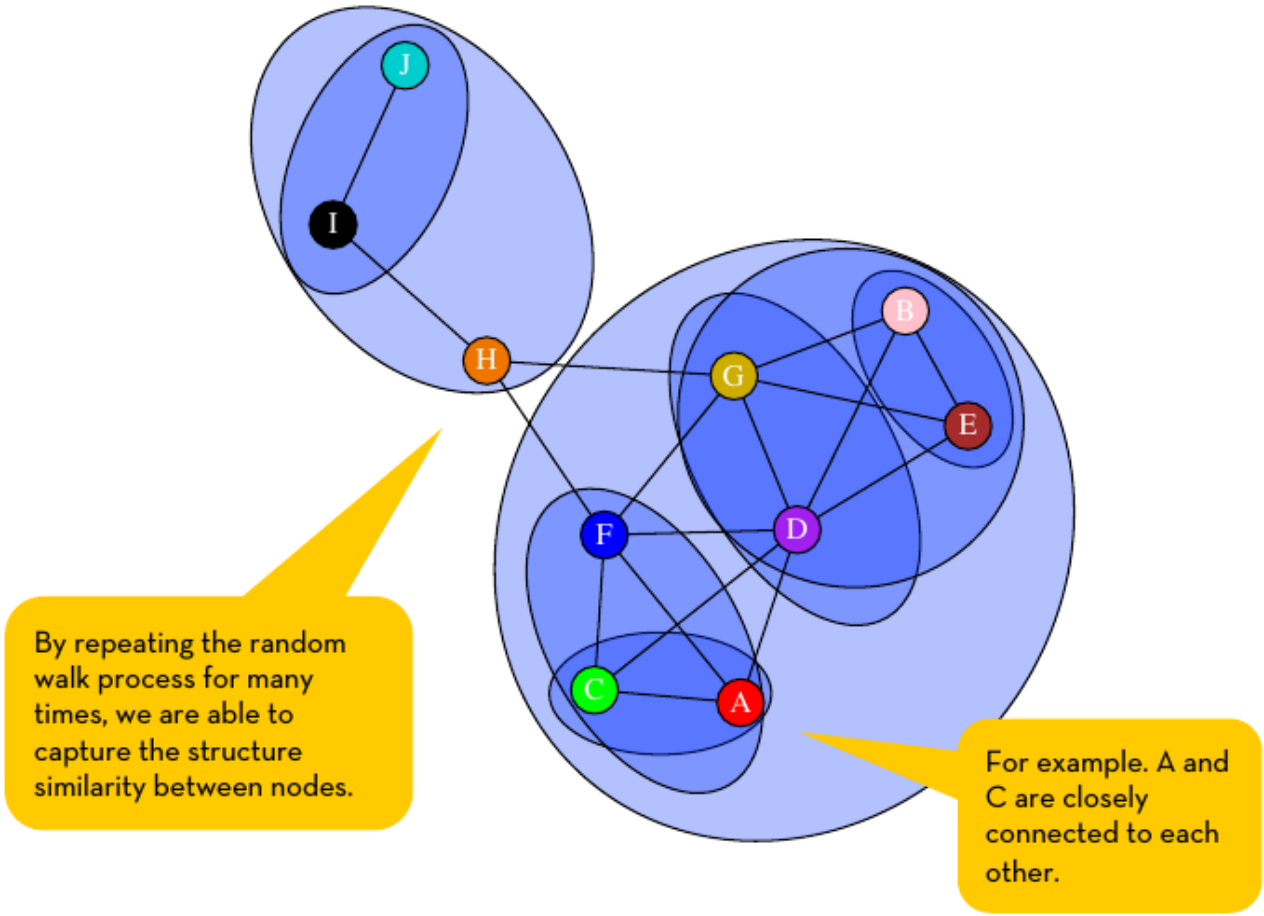


Walk trap algorithm

One random walk of length 4: C - A - D - C - F

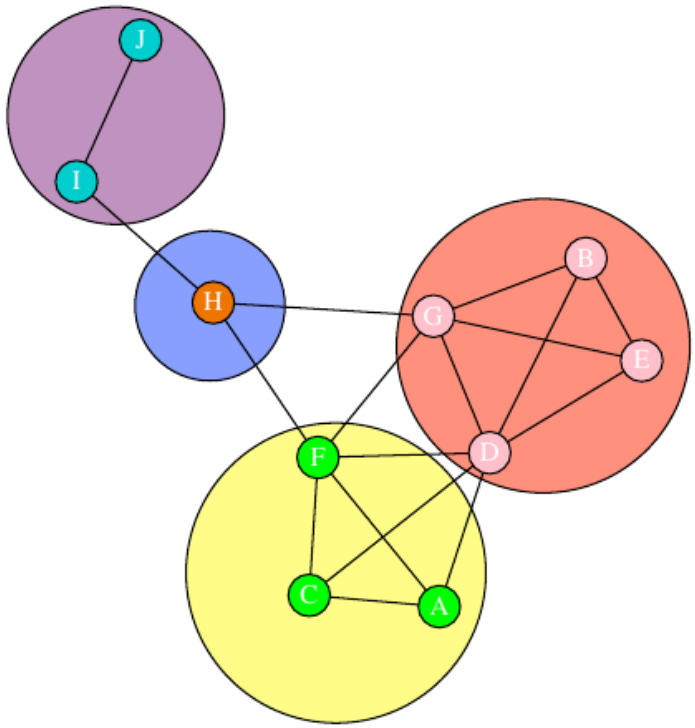


Walk trap algorithm



Walk trap algorithm: result

By merging communities in hierarchical order, find the best partition in terms of modularity score.



Walk trap algorithm: properties

- Cannot be used on large networks
- Requires large memory allocation
- Can be quite slow
- Unstuble because of intrinsic randomness
- Random walk size matters a lot

Leading Eigenvector Algorithm

Idea

Merge smaller partitions into larger ones as long as the modularity score improves

Multilevel and Fast greedy algorithm

- Detect communities by directly optimising the modularity score
- Bottom-up idea: merge communities while modularity increases
- **Assumption**: *high modularity leads to good partitions* (not necessarily true)

Multilevel algorithm

1. Assign each node to its own cluster
2. For every node:
   - 2.1 Try moving it to a community of its neighbour
   - 2.2 Calculate modularity change (no need for whole recalculation!!!)
   - 2.3 Do that for all neighbours (if consider all other communities => fast-greedy algorithm)
3. Repeat with hyper-nodes (communities) until increase in modularity becomes negative

Multilevel algorithm

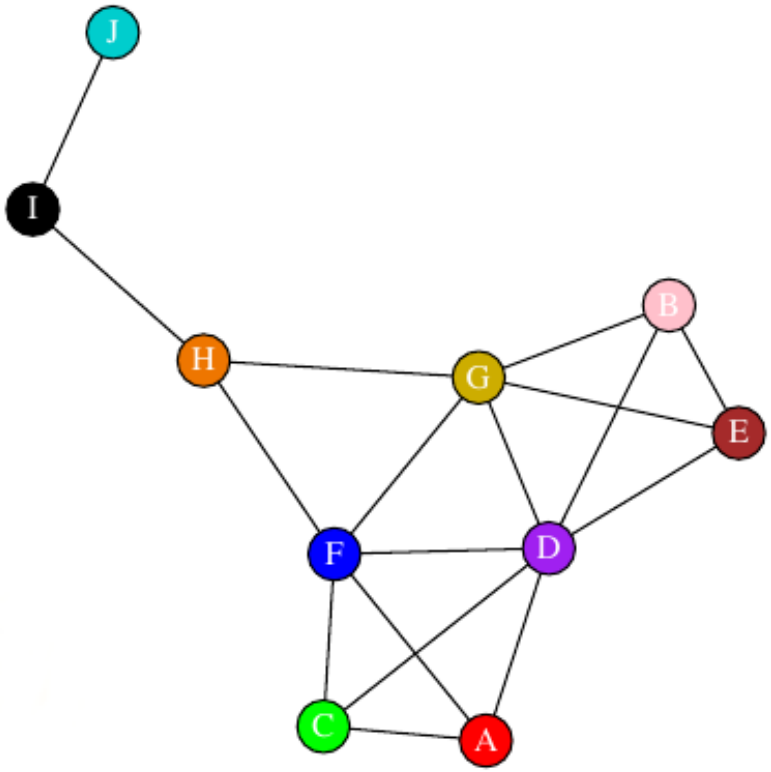
1. Assign each node to its own cluster
2. For every node:
   - **2.1** Try moving it to a community of its neighbour
   - **2.2** Calculate modularity change (no need for whole recalculation!!!)
   - **2.3** Do that for all neighbours (if consider all other communities => fast-greedy algorithm)
3. Repeat with hyper-nodes (communities) until increase in modularity becomes negative

Multilevel algorithm

1. Assign each node to its own cluster
2. For every node:
   - **2.1** Try moving it to a community of its neighbour
   - **2.2** Calculate modularity change (no need for whole recalculation!!!)
   - **2.3** Do that for all neighbours (if consider all other communities => fast-greedy algorithm)
3. Repeat with hyper-nodes (communities) until increase in modularity becomes negative

Multilevel algorithm example: initialisation

Initialise all nodes to be singleton-communities



Multilevel algorithm example: modularity change matrix

Entries represent change in modularities if the two communities are joined.

	A (1)	B (2)	C (3)	D (4)	E (5)	F (6)	G (7)	H (8)	I (9)	J (10)
A (1)	-	0	0.022	0.014	0	0.016	0	0	0	0
B (2)	0	-	0	0.014	0.022	0	0.016	0	0	0
C (3)	0.022	0	-	0.014	0	0.016	0	0	0	0
D (4)	0.014	0.014	0.014	-	0.014	0.003	0.003	0	0	0
E (5)	0	0.022	0	0.014	-	0	0.016	0	0	0
F (6)	0.016	0	0.016	0.003	0	-	0.008	0.016	0	0
G (7)	0	0.016	0	0.003	0.016	0.008	-	0.016	0	0
H (8)	0	0	0	0	0	0.016	0.016	-	0.024	0
I (9)	0	0	0	0	0	0	0	0.024	-	0.028
J (10)	0	0	0	0	0	0	0	0	0.028	-

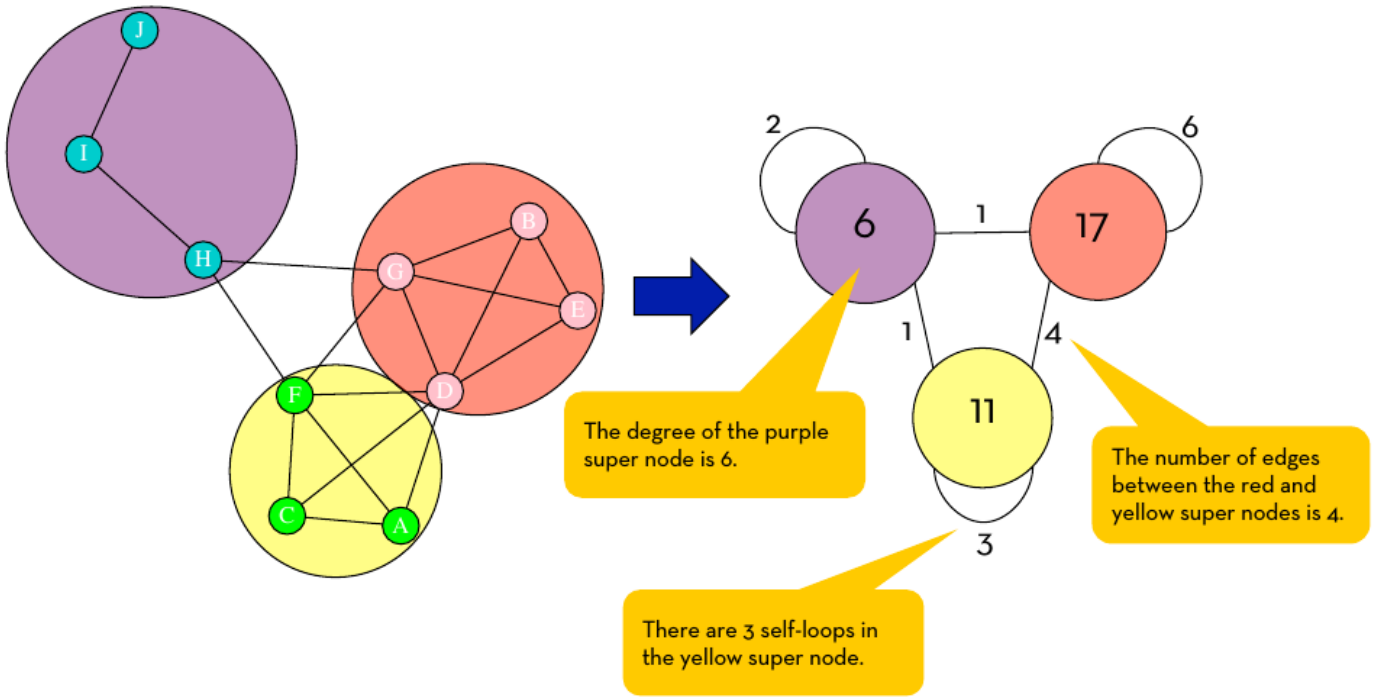
Multilevel algorithm example: modularity change matrix

Find maximal value for every row and form new communities (ties broken randomly)

	A (1)	B (2)	C (3)	D (4)	E (5)	F (6)	G (7)	H (8)	I (9)	J (10)
A (1)	-	0	0.022	0.014	0	0.016	0	0	0	0
B (2)	0	-	0	0.014	0.022	0	0.016	0	0	0
C (3)	0.022	0	-	0.014	0	0.016	0	0	0	0
D (4)	0.014	0.014	0.014	-	0.014	0.003	0.003	0	0	0
E (5)	0	0.022	0	0.014	-	0	0.016	0	0	0
F (6)	0.016	0	0.016	0.003	0	-	0.008	0.016	0	0
G (7)	0	0.016	0	0.003	0.016	0.008	-	0.016	0	0
H (8)	0	0	0	0	0	0.016	0.016	-	0.024	0
I (9)	0	0	0	0	0	0	0	0.024	-	0.028
J (10)	0	0	0	0	0	0	0	0	0.028	-

Multilevel algorithm example: second iteration

Do the same for the resulting partition considering communities as super-nodes



Multilevel algorithm example: second iteration

All potential changes in modularity score are negative, so we stop the algorithm.

	Purple	Yellow	Red
Purple	-	-0.94	-2
Yellow	-0.94	-	-1.5
Red	-2	-1.5	-

Multilevel algorithm properties

- Tends to form large communities at the expenses of small ones
- There exists resolution limit for modularity: it does not work well for large graphs

Algorithms comparison

Approach to algorithm-testing

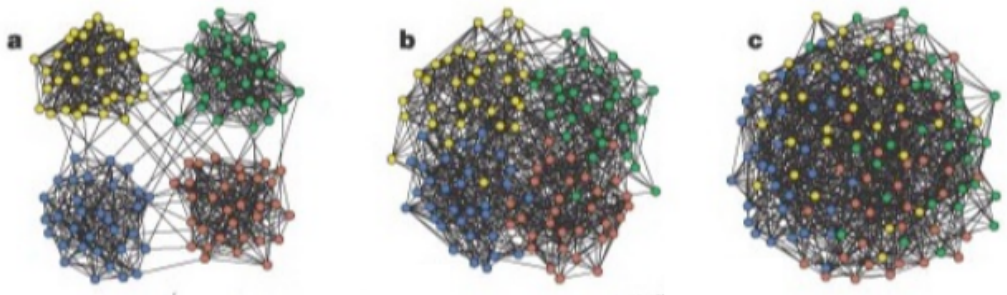
- We need to compare the algorithms' performance
- We apply them to the task, where the correct solution is known *a priori*.
- There are two setups where it is possible:
  - Synthetic data
  - Real-world data with clear community structure

Different benchmarks are introduced to produce reproducible and well-controlled environment for experimenting with community detection algorithms

Synthetic data: GN Benchmark

[Guimerà,Nunes Amaral, *Functional cartography of complex metabolic networks* (2005)]

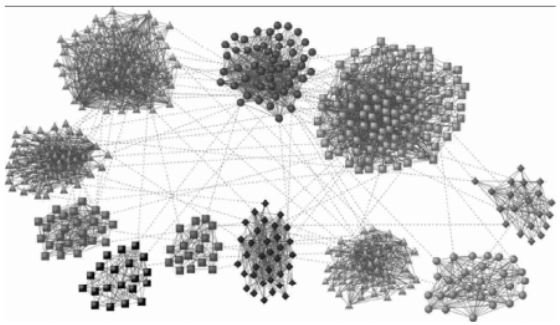
- 4×32-nodes subgraphs
- Fixed average degree is 16
- Tuning parameters: internal and external degrees
- Cases a, b, c correspond to expected internal degree of 15, 11, 8 respectively



Synthetic data: LFR Benhmark

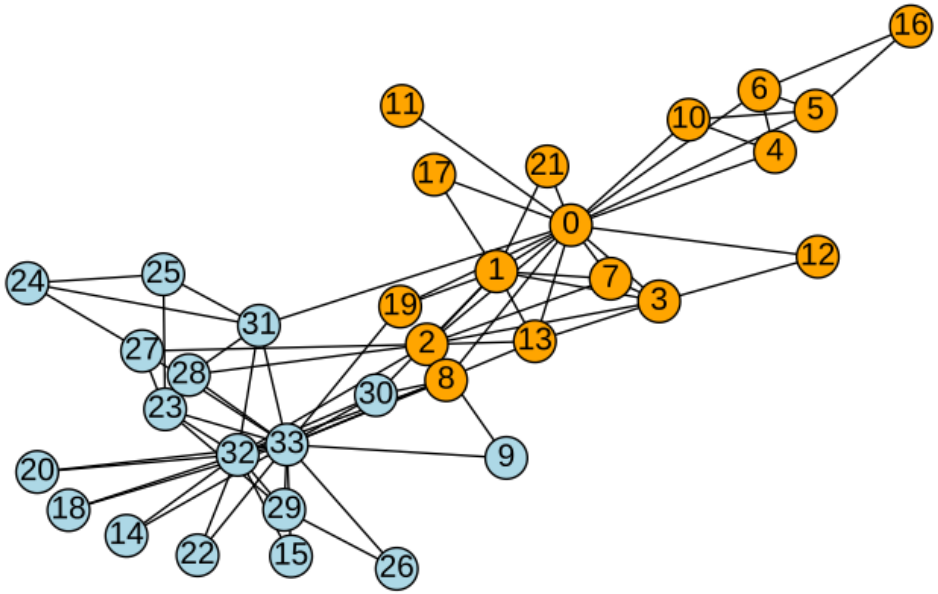
[Lancichinetti, Fortunato, Radicchi, *Benchmark graphs for testing community detection algorithms* (2008)]

- Extension of GN benchmark
- Degree distribution follows power law
- Control parameters:
  - average and maximum degree
  - minimum, average, maximum community size
  - Mixing parameter μ: Controls the proportion of links across communities



Real benchmarks

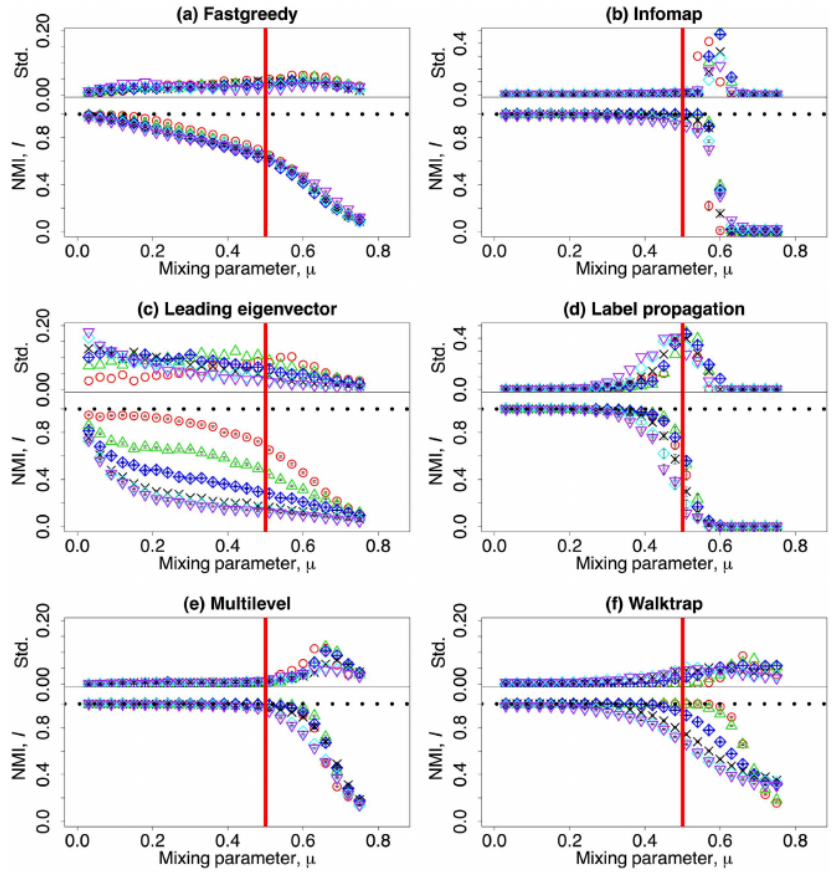
- **Classic example: Zachary's karate club network**
- Network of social relationships between members of a karate club in the US
- Following a disagreement between the instructor (node 0) and the president (node 33), the club split in two different groups
- The task is to identify those groups (identified by the node colors) using a clustering algorithm

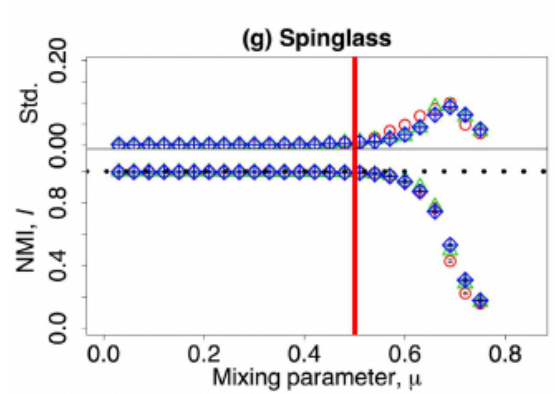


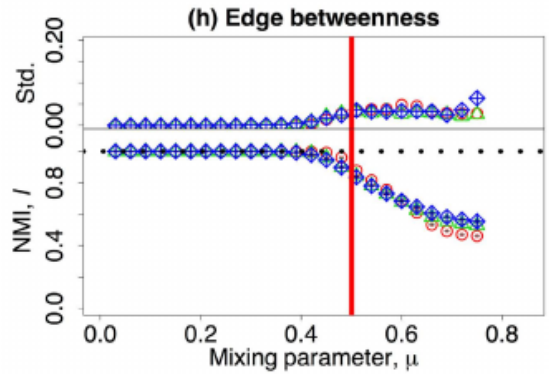

```
G = nx.karate_club_graph()
```

Performance on LFR benchmarks

[Z. Yang, R. Algesheimer, C.J. Tessone, *A Comparative Analysis of Community Detection Algorithms* ... (2016)]

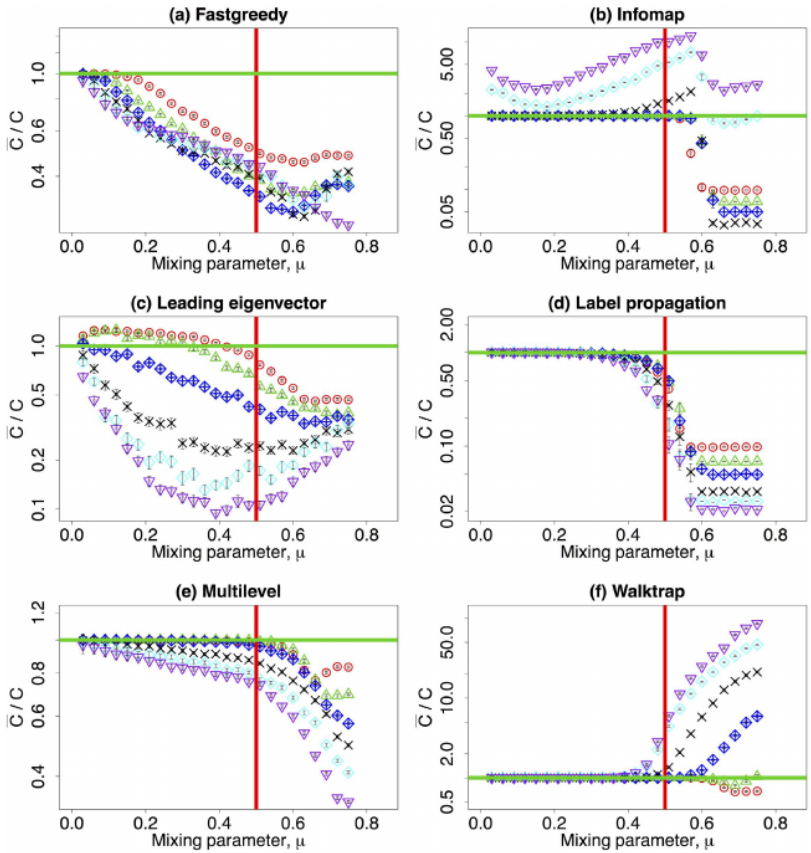


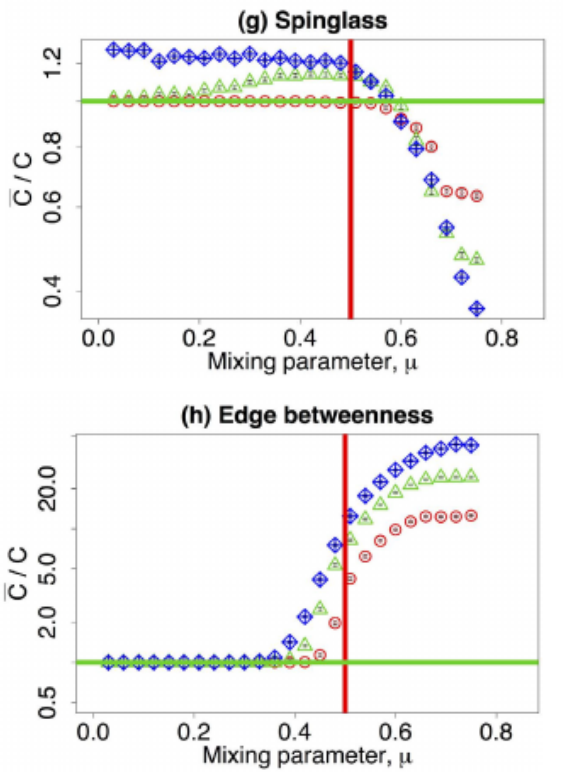




Time on LFR benchmarks

[Z. Yang, R. Algesheimer, C.J. Tessone, *A Comparative Analysis of Community Detection Algorithms* ... (2016)]





Algorithms overview

	Algorithm	Stable	Fast
Modularity based	Leading eigenvector	No	No
	Fast greedy	Yes	Yes
	Multilevel	**Yes**	**Yes**
Random walk	Walktrap	Yes	No
Other	Edge betweenness	Yes	No
	Label propagation	No	Yes

References I

- Tiago P. Peixoto, *Hierarchical block structures and high-resolution model selection in large networks*, in Physical Review X, 2014.
- Santo Fortunato, *Community detection in graphs*, 2010.
- Aaron Clauset, M. E. J. Newman, Cristopher Moore, *Finding community structure in very large networks*, In Phys. Rev, 2004.
- M. E. J. Newman, *Modularity and community structure in networks*, In PNAS, 2006.

References II

- Vincent D. Blondel, Jean-Loup Guillaume, Renaud Lambiotte, Etienne Lefebvre, *Fast unfolding of communities in large networks*, In J. Stat. Mech, 2008.
- Pascal Pons, Matthieu Latapy, *Computing communities in large networks using random walks*, 2005
- M. Girvan and M. E. J. Newman, *Community structure in social and biological networks*, In PNAS, 2002.
- Usha Nandini Raghavan, Reka Albert, Soundar Kumara, *Near linear time algorithm to detect community structures in large-scale networks*, In Physical Review, 2007.

References III

- Roger Guimera , Luis A. Nunes Amaral, *Functional cartography of complex metabolic networks*, In Nature, 2005.
- Andrea Lancichinetti, Santo Fortunato, Filippo Radicchi, *Benchmark graphs for testing community detection algorithms*, In Physical Review, 2008
- Yang, Z., Algesheimer, R., Tessone, C. J., *A comparative analysis of community detection algorithms on artificial networks*, In Scientific reports, 6, 30750, 2016.

Claudio J. Tessone

Blockchain & Distributed Ledger Technologies
UZH Blockchain Center

claudio.tessone@uzh.ch
http://www.ifi.uzh.ch/bdlt