UNIVERSITY OF MANNHEIM

Web Mining

Web Structure Mining and Social Network Analysis

- Part 2 -

Prof. Dr. Christian Bizer

FSS 2025

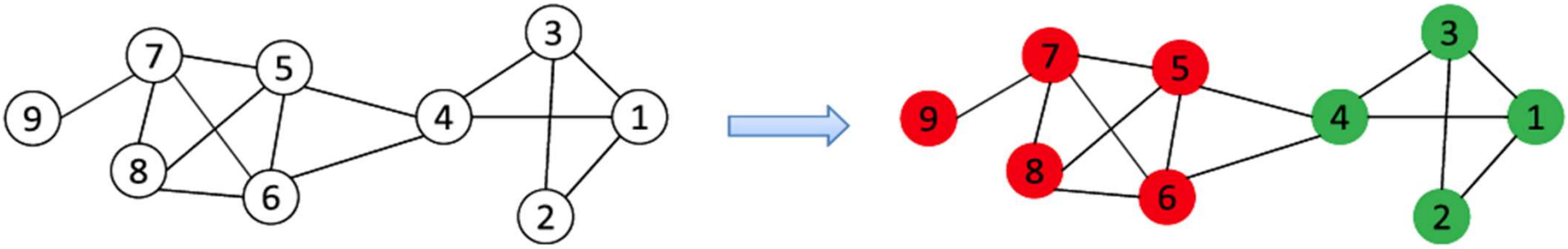
Chapter Outline

1. Describing Graphs
   1. Terminology and Measures
2. Prominence
   1. Centrality
   2. Prestige
3. Community Detection
   1. Connected Components and K-Cores
   2. Clustering-based Techniques
4. Machine Learning with Graphs
   1. Link Prediction and Node Classification
   2. Node Embeddings
   3. Graph Neural Networks

3. Community Detection

A community is a set of actors between which interactions are (relatively) frequent.

- Communities are also called groups, cohesive subgroups, clusters, or modules in different contexts
- Finding a community in a social network is to identify a set of nodes such that they interact with each other more frequently than with those nodes outside the group



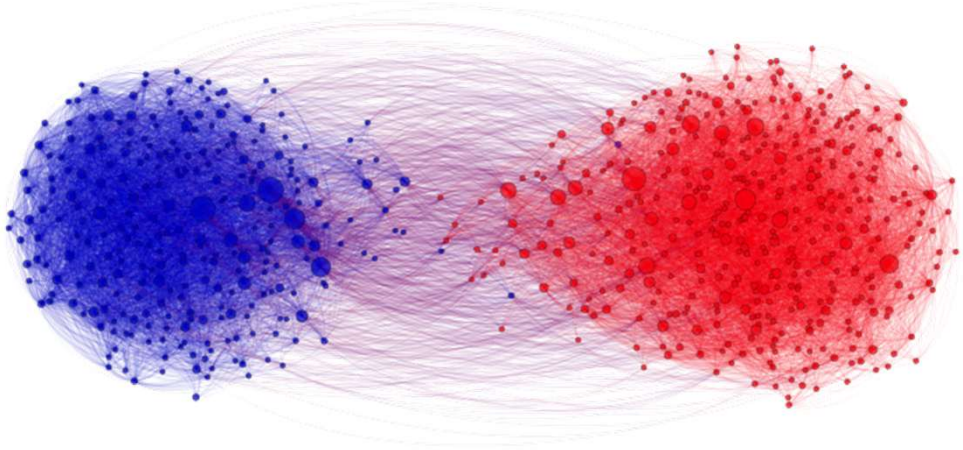
Why Analyze Communities?

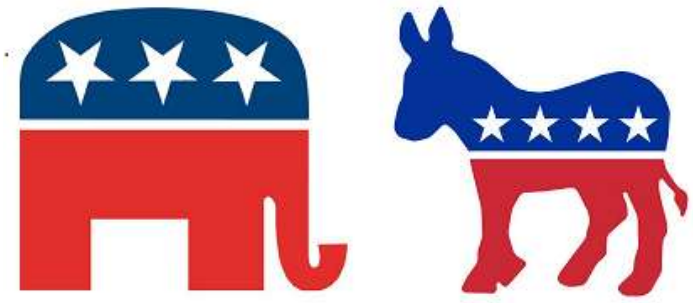
Analyzing communities helps to better understand users

- patters might be clearer on group level

Groups provide a clear global view of user interactions

- network compression
- visualization of huge networks





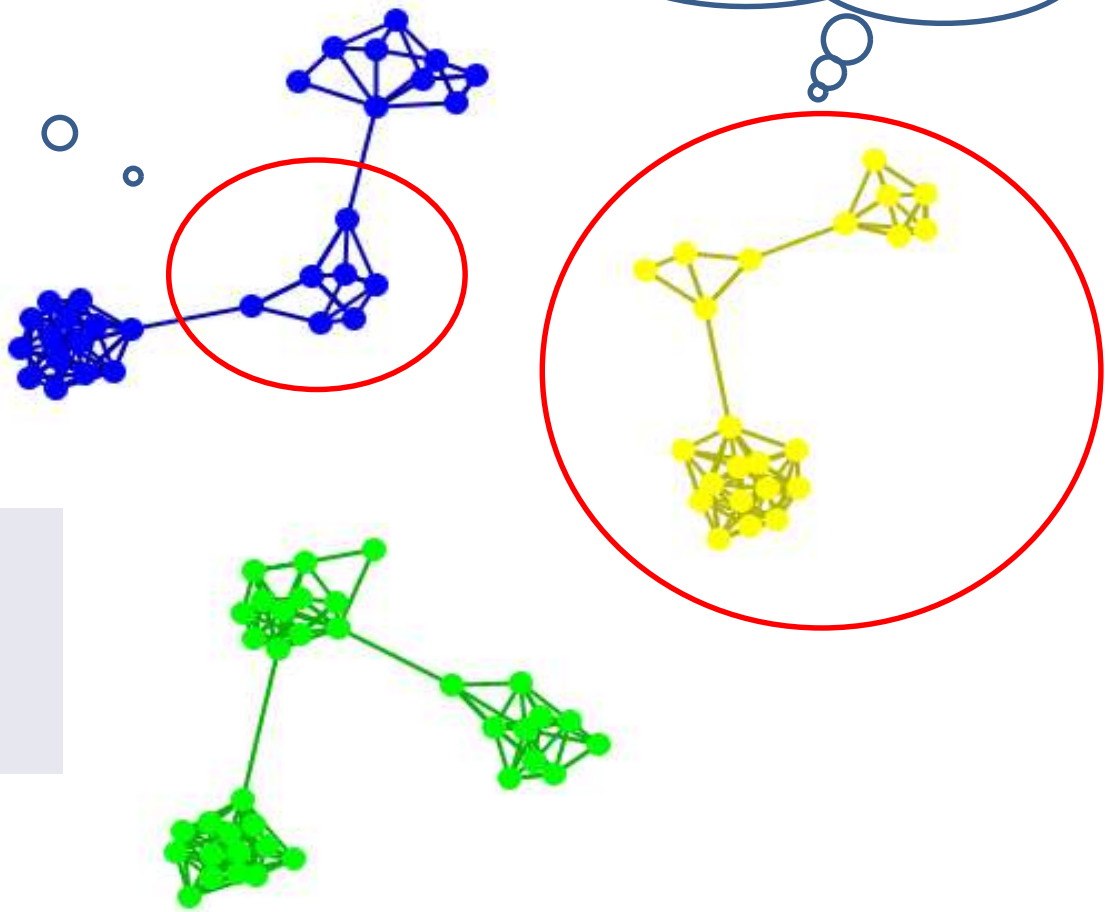
Comparing individual behavior and behavior expected by the group might lead to interesting insights

- some republican can agree with some democrats, but their parties disagree

Task-Dependence of Community Definition

A densely-knit community?

Each component is a community?

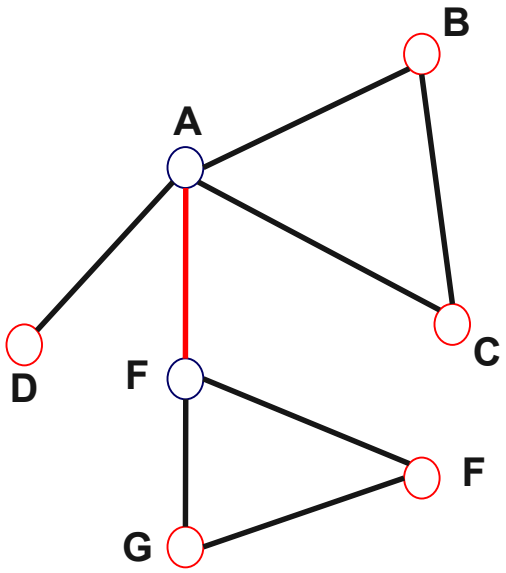


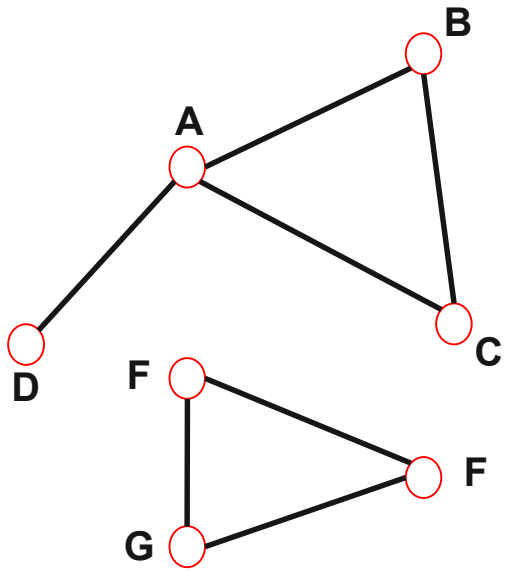
The perception of community is task dependent

3.1 Graph Connectivity and Components

Connected graph: Any two vertices of an undirected graph are joined by a path.

A disconnected graph is made up by two or more connected components.

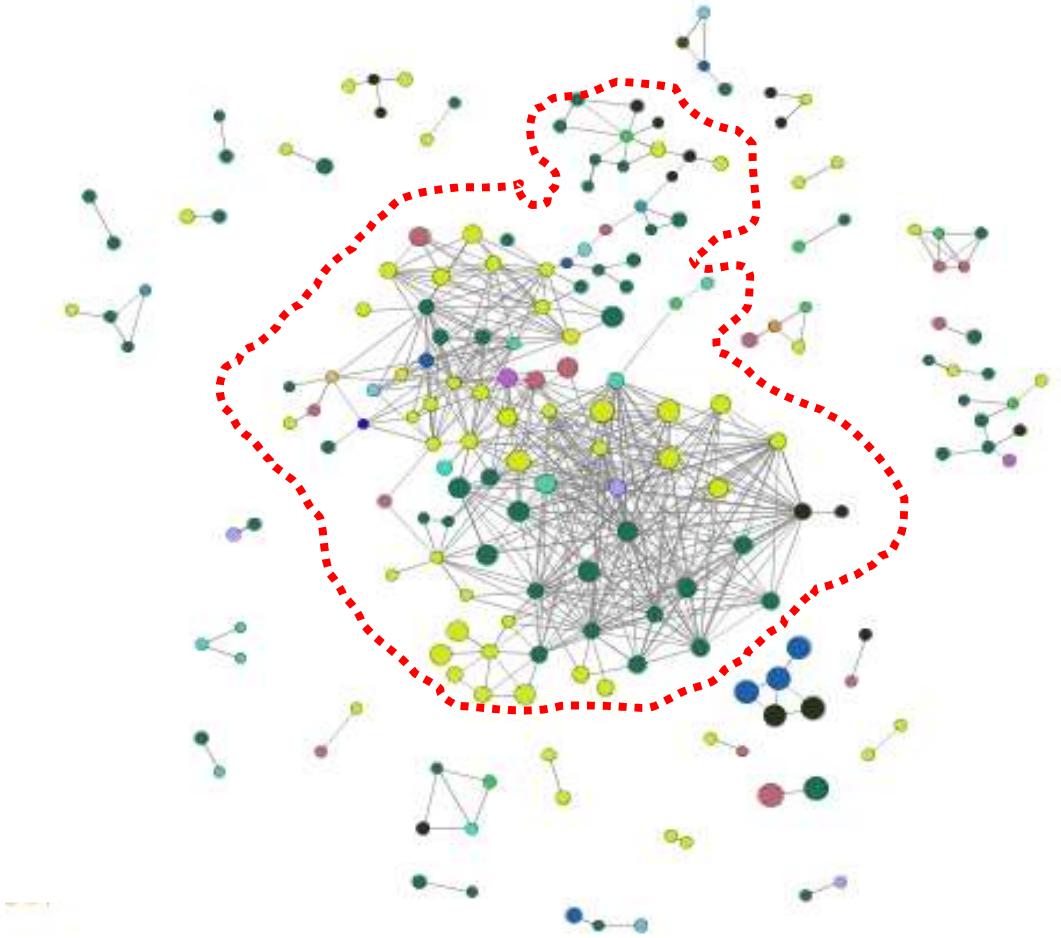




Bridge: if we remove it, the graph becomes disconnected (red line).
Cut vertex: if we remove it, the graph becomes disconnected (F and A).

Giant Component and Isolates

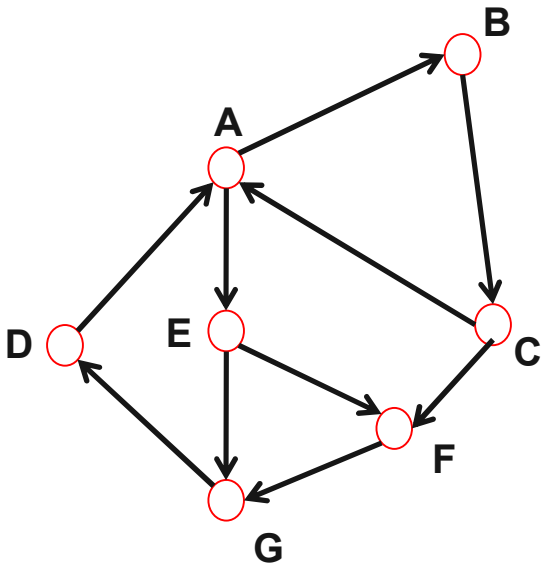
- If the largest component encompasses a significant fraction of the graph, it is called the giant component.
- The other smaller components are called isolates.



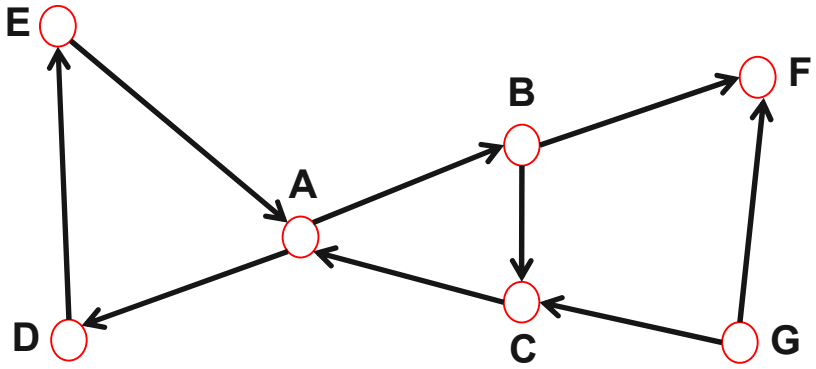
Connectivity of Directed Graphs

Strongly connected directed graph: has a path from each vertex to every other vertex and vice versa (e.g. AB path and BA path).

Weakly connected directed graph: is connected if we disregard the arc directions.



Strongly connected directed graph



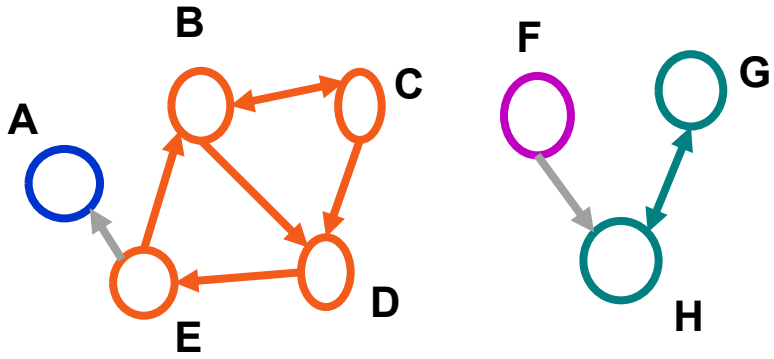
Weakly connected directed graph

Connected Components of Directed Graphs

Strongly Connected Component (SCC)

Maximal subgraph in which every vertex can be reached from every other vertex by following directed arcs. Interesting for web crawling.

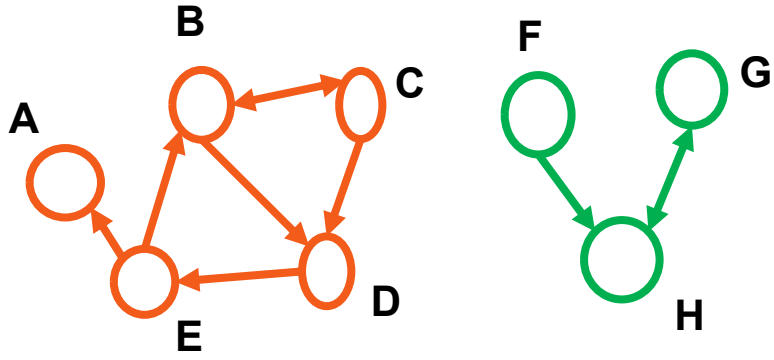
- Strongly connected components
  - B C D E
  - A
  - G H
  - F



Weakly Connected Component (WCC)

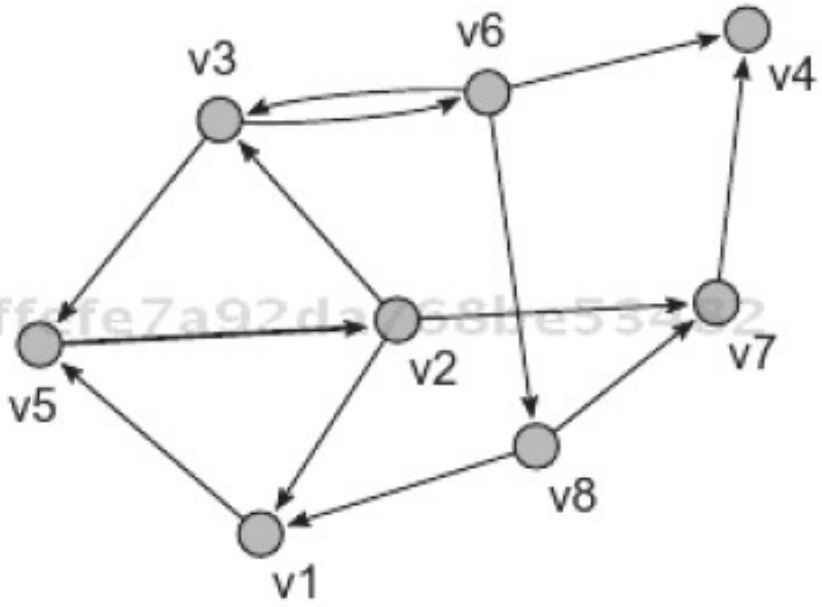
Maximal subgraph in which every vertex can be reached from every other vertex by following lines in either direction.

- Weakly connected components
  - A B C D E
  - G H F



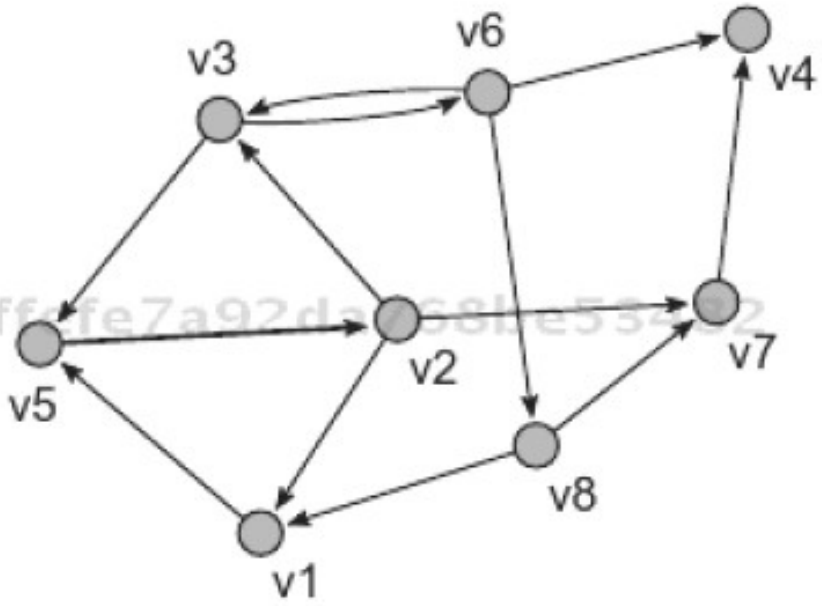
Exercise: Components

- How many strongly connected components has the graph below?



Solution: Components

- How many strongly connected components has the graph below?



- Answer: 3 components
  1. v4
  2. v7
  3. and rest of graph

Largest Strongly Connected Component

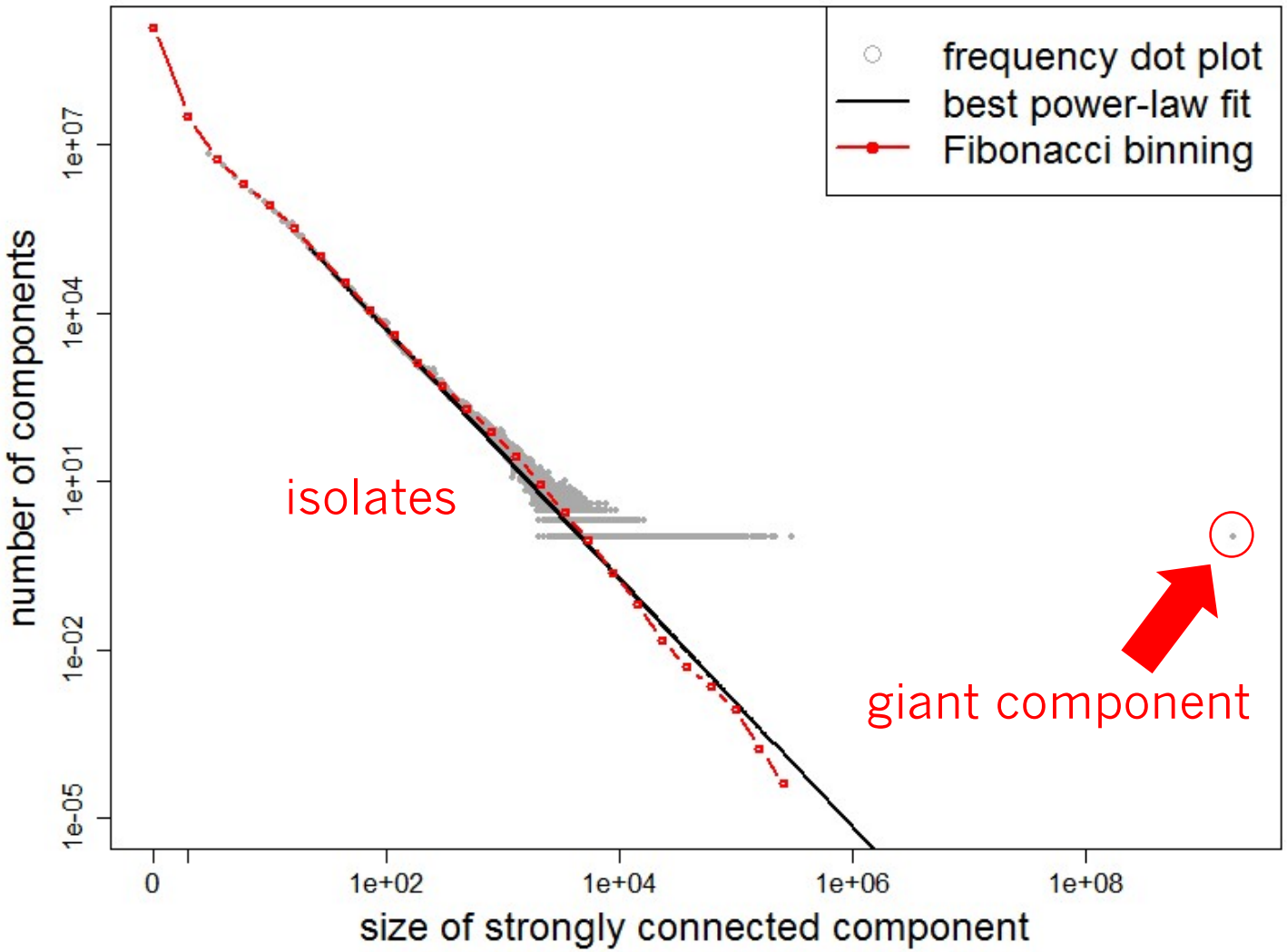
Largest SCC

- Broder, 2000: 27.7%
- WDC, 2012: 51.3 %

➔ Factor 1.8 larger

➔ Also, factor 4.9 more links/page

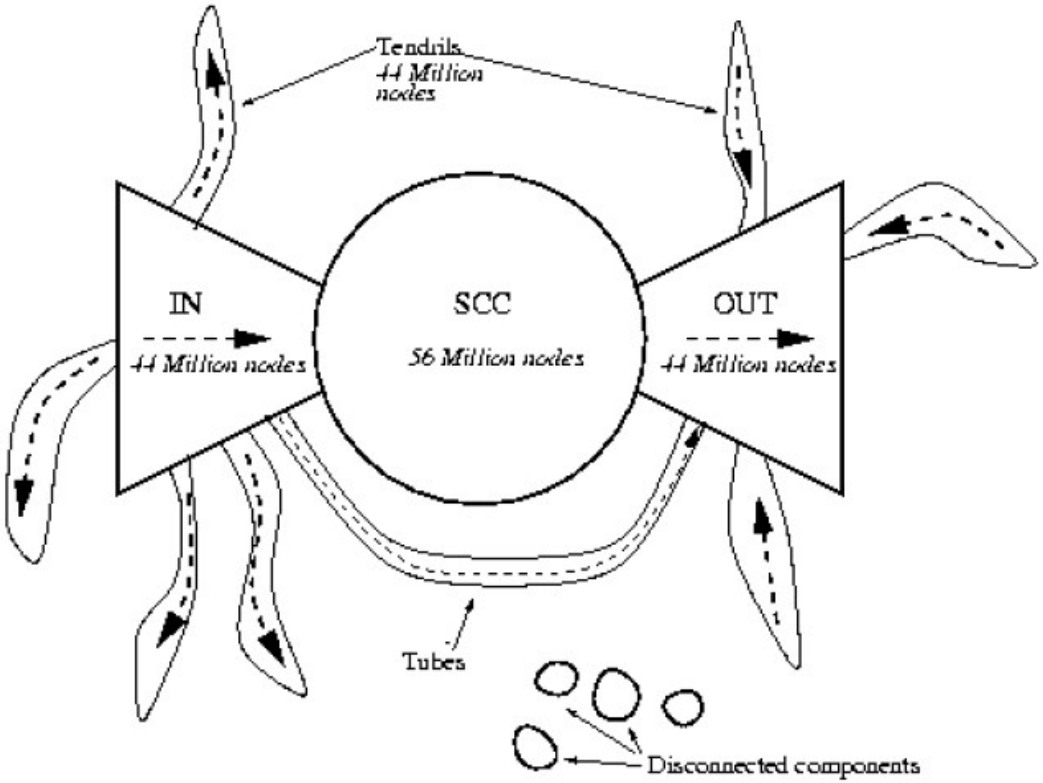
➔ The Web has become denser.



Hyperlink Structure of the Web: The Bow-Tie

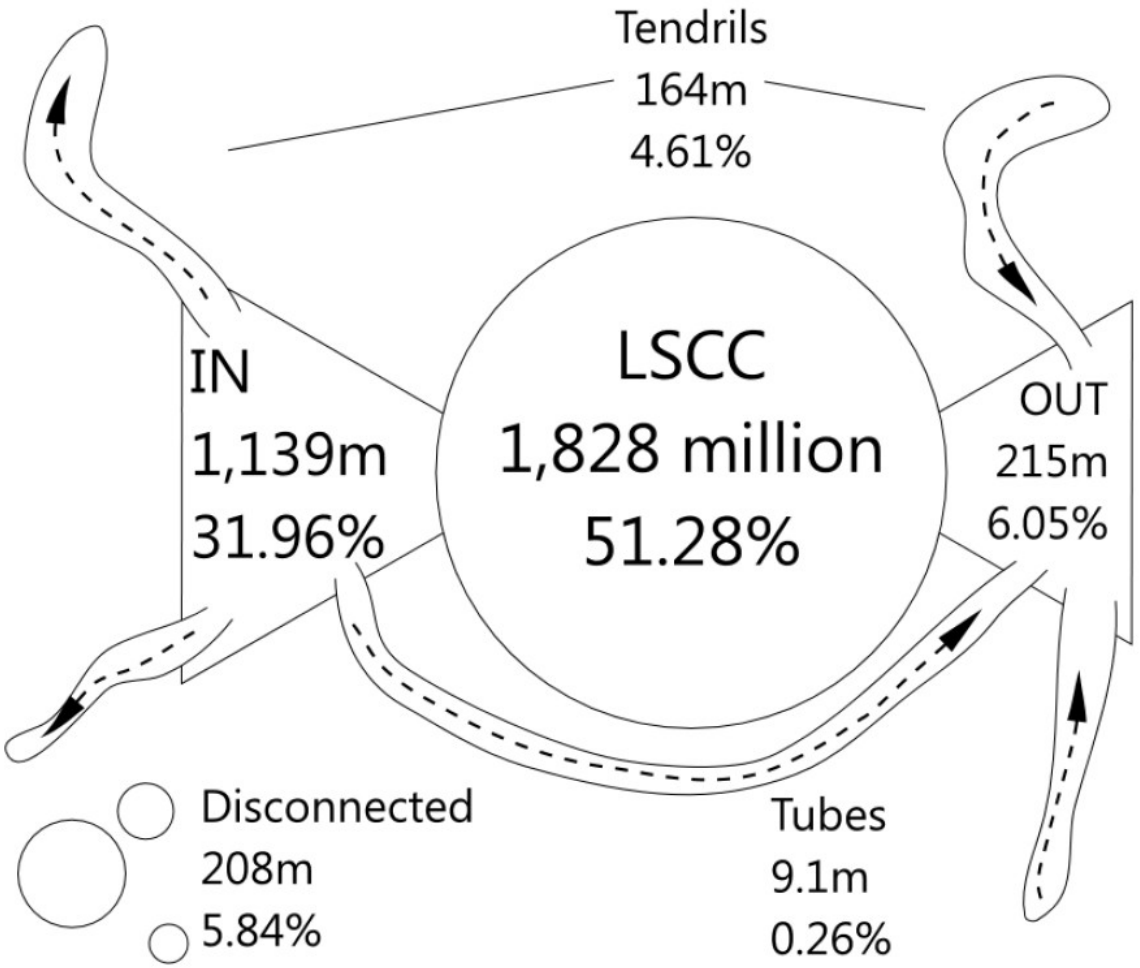
Four mayor parts (Broder et al., WWW2000)

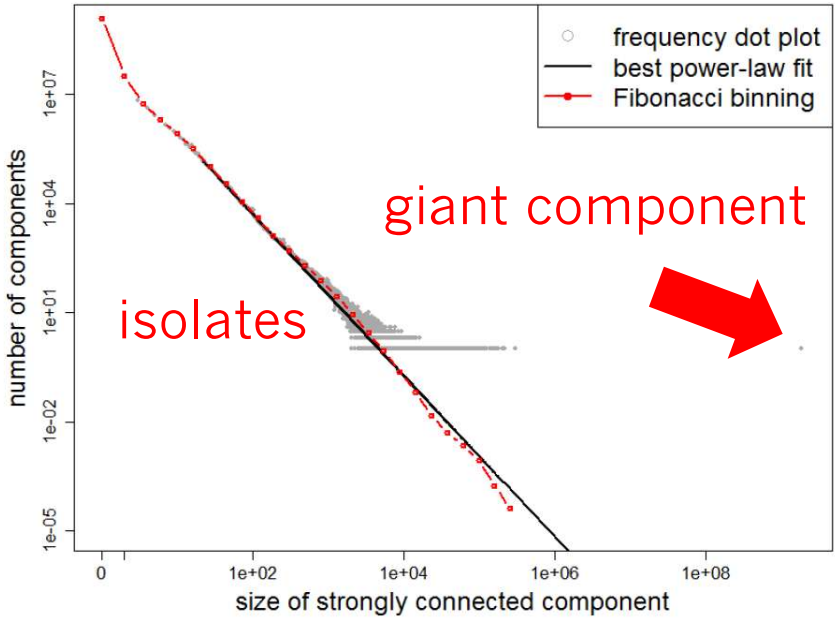
- **Central Strongly Connected Component (SCC)**
  - pages that can reach one another along hyperlinks
  - about 30% of the Web (normal pages)
- **IN Group**
  - can reach SCC via directed path but cannot be reached from it
  - about 20% (maybe new pages or boring ones)
- **OUT Group**
  - can be reached from SCC but cannot reach it
  - about 20% (maybe company pages that don't link)
- **Tendrils**
  - cannot reach SCC and cannot be reached by it
  - about 20%
- **Unconnected**
  - about 10%



Probability of a path between 2 nodes is 24%

Bow-Tie and SCCs in WDC Hyperlink Graph





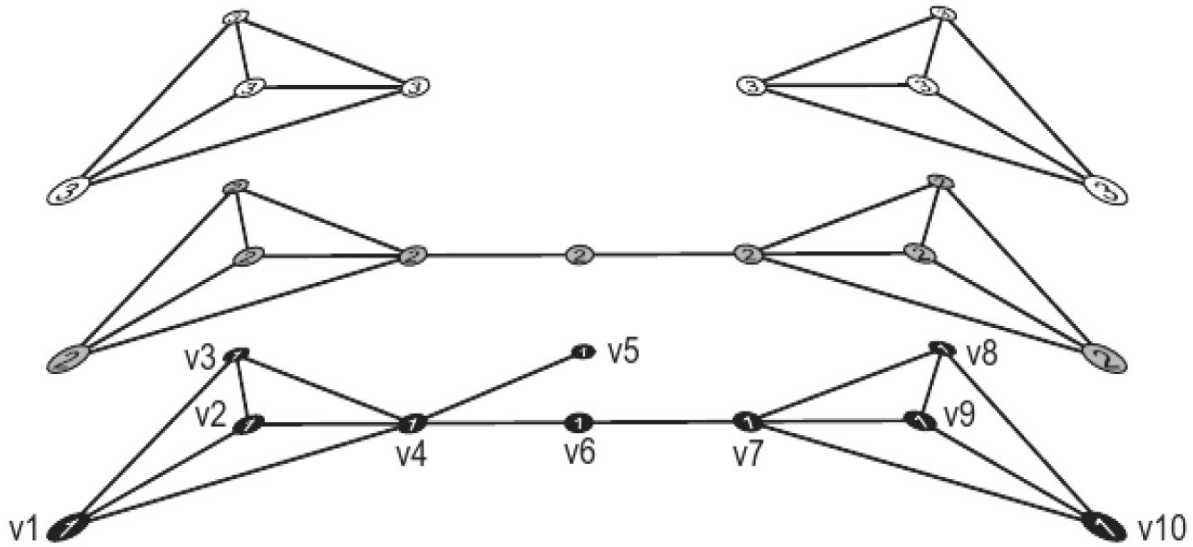
Probability of a path between 2 nodes is 48%

- IN much larger than OUT: 31% vs. 6%
- LSCC much larger compared to Broder et al.: 51% vs. 30%
- The Web has become more connected in this 14 years

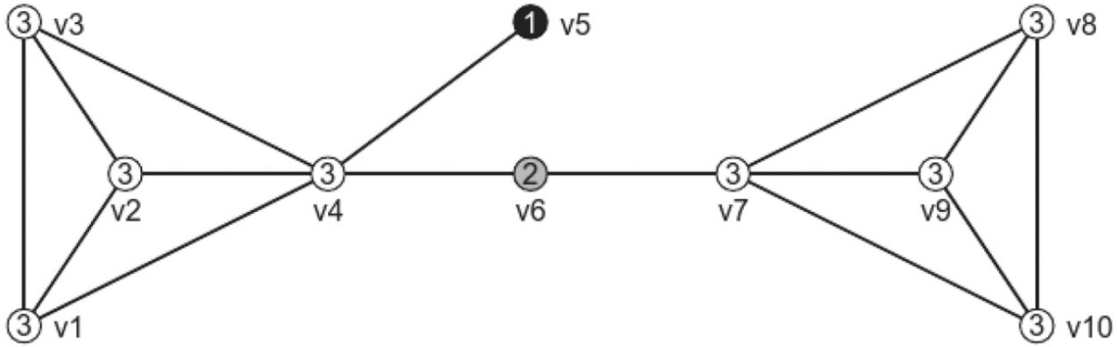
Meusel, Vigna, Lehmberg, Bizer: Graph Structure in the Web - Revisited. 23rd Conference on World Wide Web (WWW2014).

3.2 K-Cores

A K-Core is a maximal undirected subgraph in which each vertex has at least degree k within the subgraph.

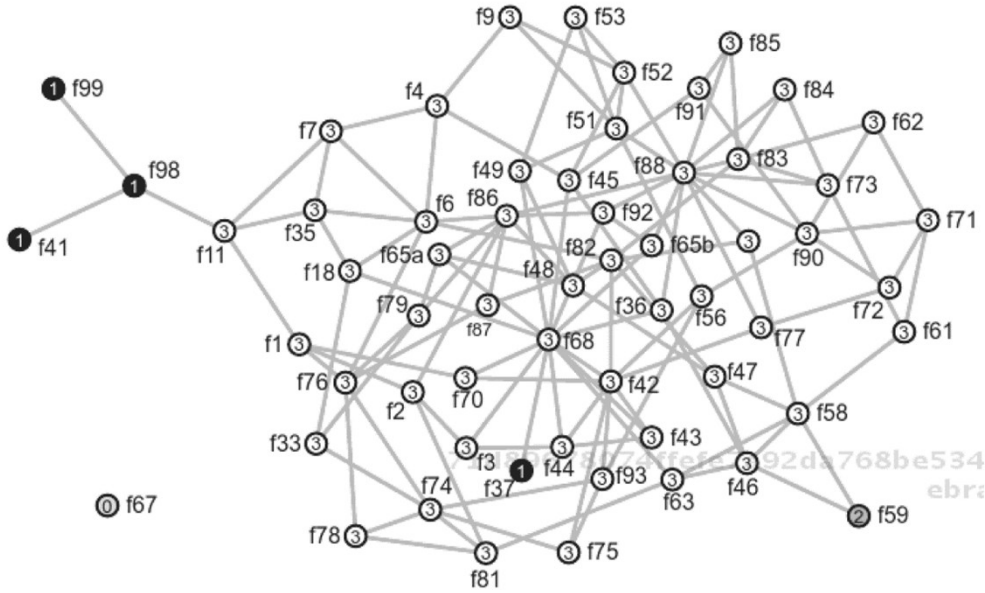


- A k-core does not need to be connected. The one 3-core above consists of two parts.
- K-cores are nested, meaning every vertex in the 3-core also belongs to the 2-core.

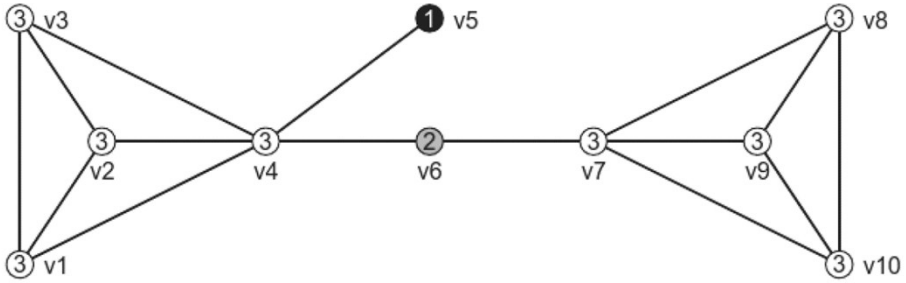


K-Cores and Communities

- A vertex is displayed belonging to the highest k-core of which it is a part



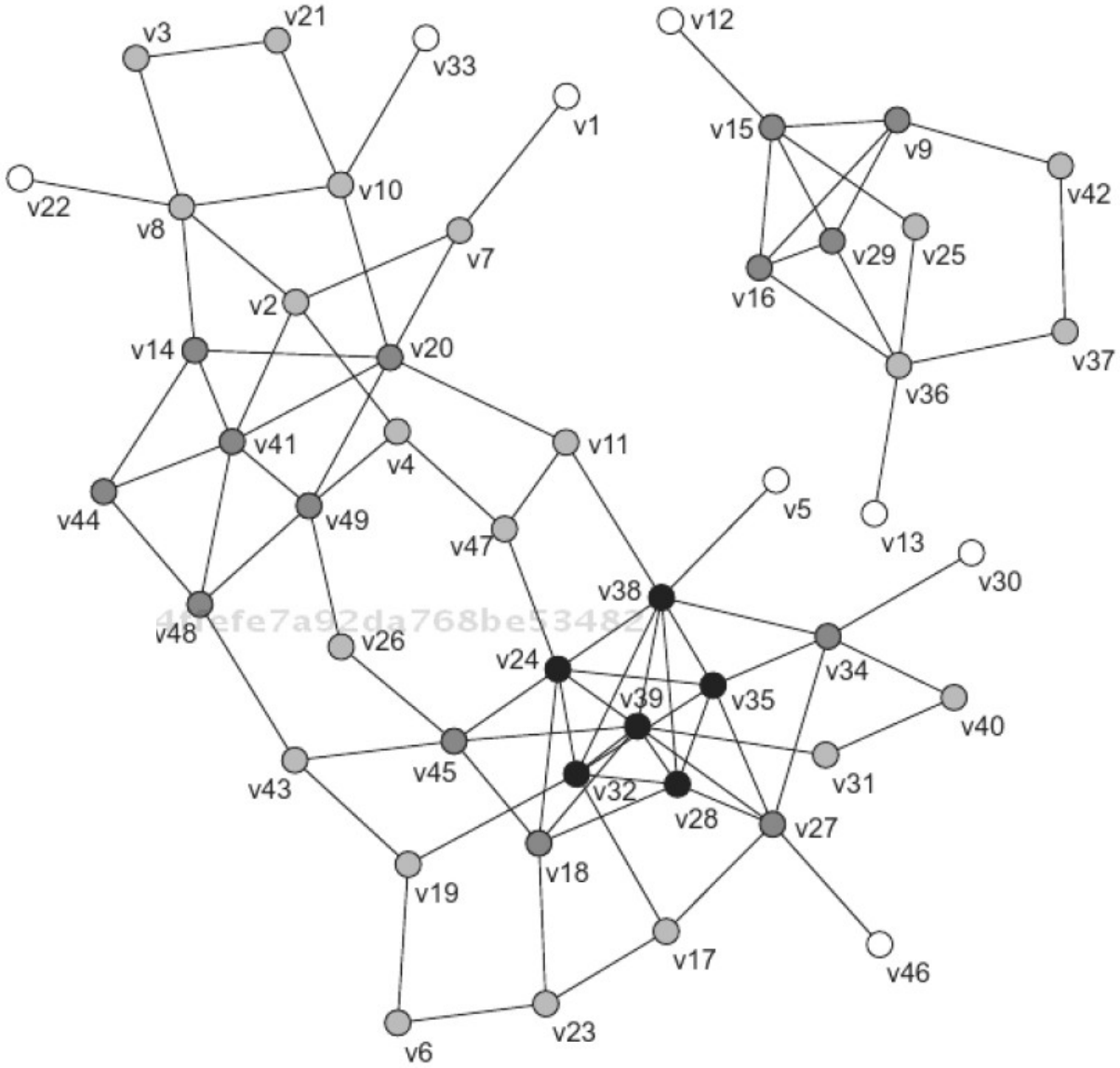
- **Finding communities:** Remove lower cores until the graph breaks up into multiple components that form cohesive subgroups



- K-cores with k>1 pose a stricter cohesion requirement than components

Exercise: K-Cores

- **Count the number of 3-cores in the graph.**



Vertex colors indicate the level of k:

White: k=1

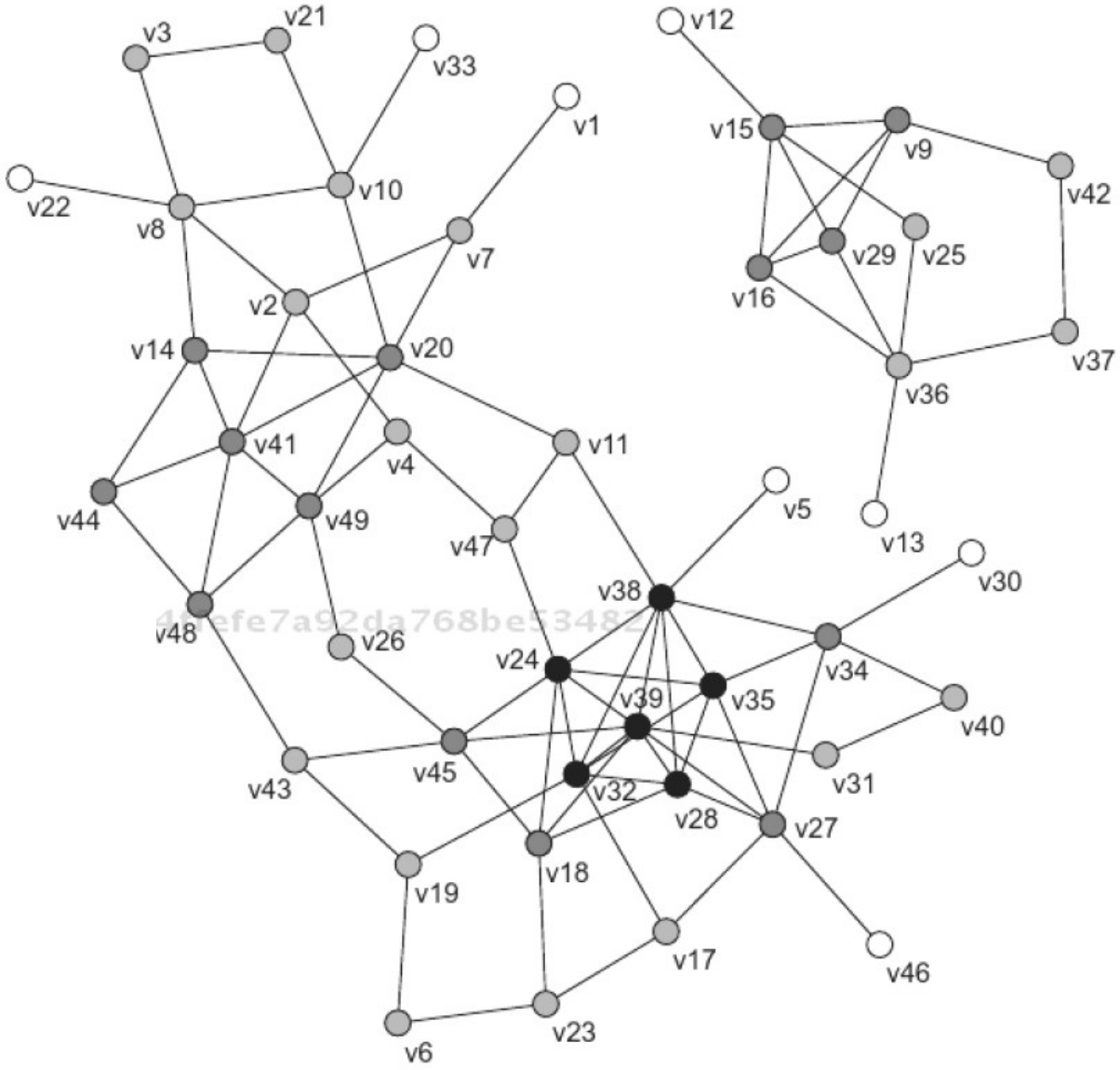
Light gray: k=2

Dark gray: k=3

Black: k=4

Solution: K-Cores

- **Count the number of 3-cores in the graph.**
- **Answer:**
  - one 3-core, as there is always only one.
  - if we remove the 1-core and 2-core, we get three components.



Vertex colors indicate the level of k:

White: k=1

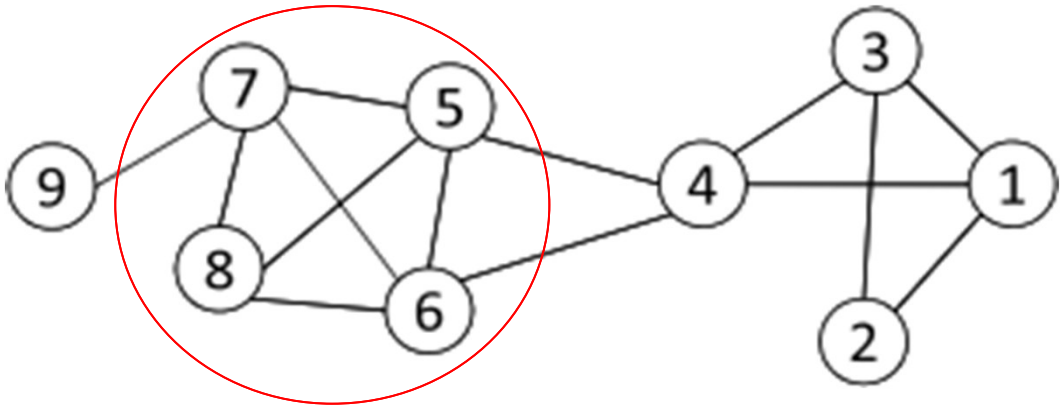
Light gray: k=2

Dark gray: k=3

Black: k=4

3.3 Cliques

A clique is the maximal complete subgraph containing three vertices or more.

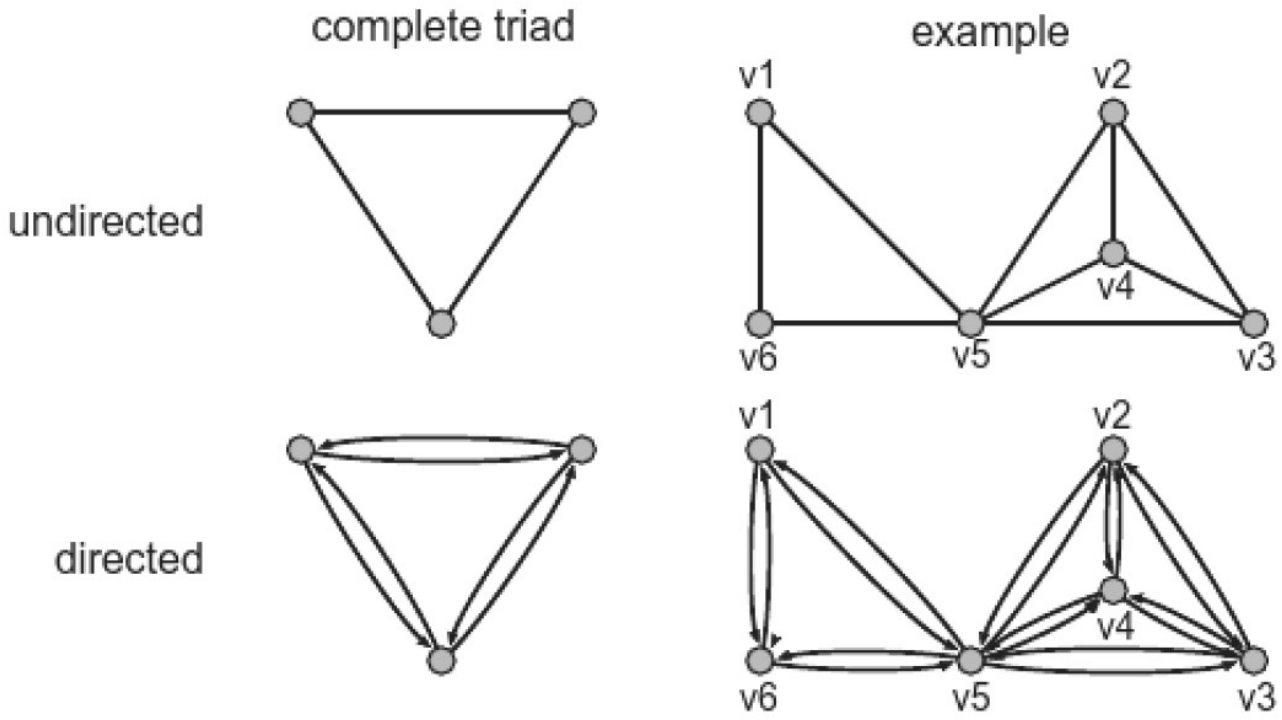


Nodes 5, 6, 7 and 8 form a clique

- Cliques are the strongest form of community as all vertices need to be directly adjacent to each other.
- Problems:
  - The definition is often too restrictive to detect communities in real-world networks (result: large number of very small cliques)
  - Finding cliques in larger graphs is computationally expensive

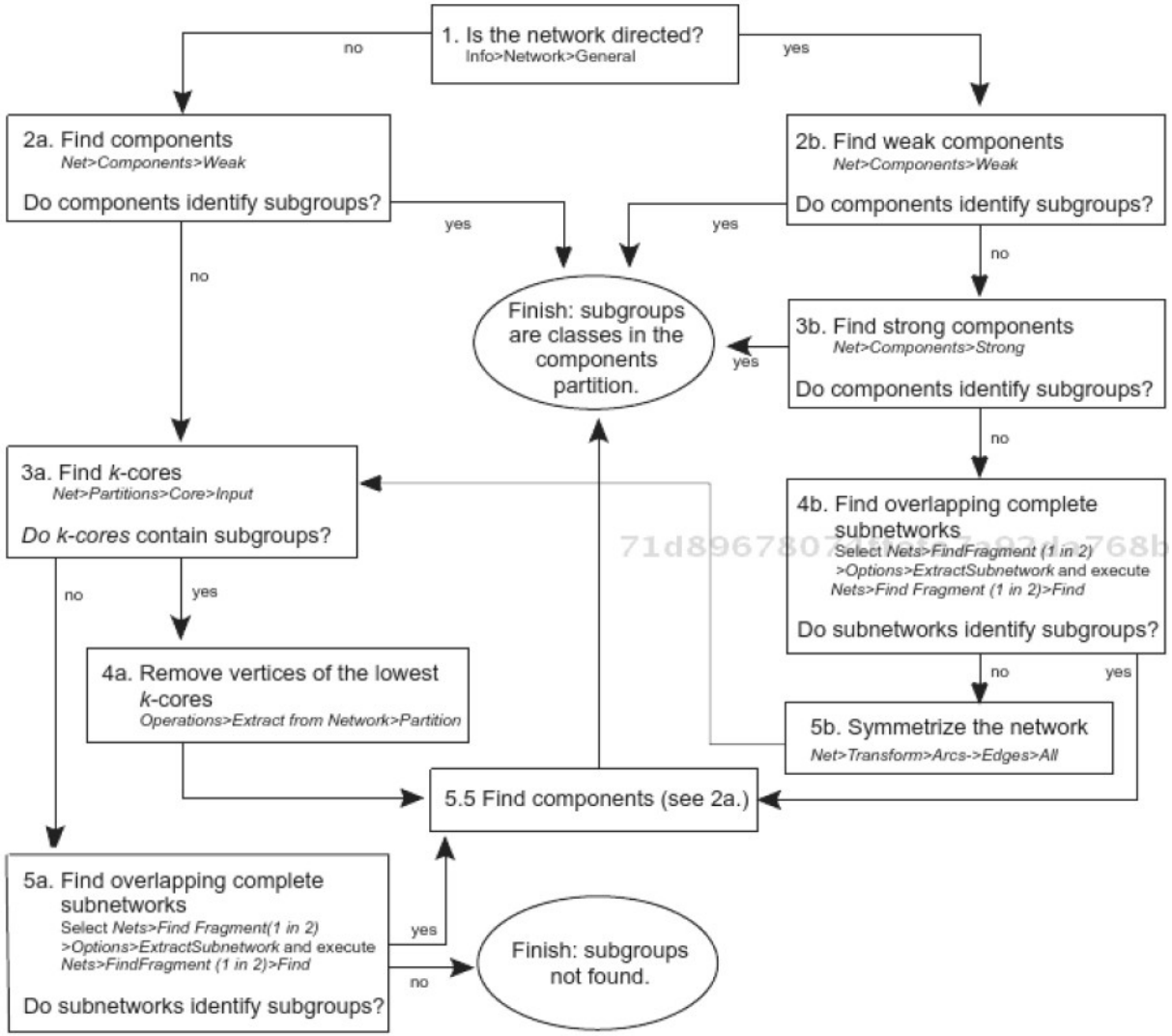
Overlapping Complete Subgraphs

- Alternative less restrictive approach: Consider overlapping complete subgraphs (triads) as cohesive group.
- Triads overlap if they share one or more vertices.



Procedure for the Analysis of Cohesive Subgroups

- Idea: Successively increase the minimal density required.
- After each step: Check if discovered groups make sense based on your knowledge about the application domain.



Hierarchical Communities

- **Previous methods consider communities at a single level**
  - Communities may form hierarchies
    - Each community can have sub/super communities
  - Hierarchical clustering deals with this scenario and generates community hierarchies

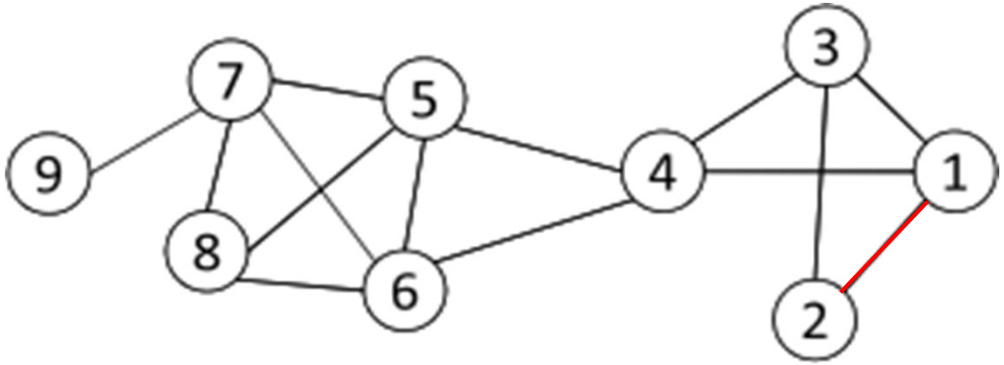
- **Initially n members are considered as either 1 or n communities in hierarchical clustering.**
- **These communities are gradually**
  - merged (agglomerative hierarchical clustering) or
  - split (divisive hierarchical clustering)

Divisive Hierarchical Clustering

- **Goal: Build a hierarchical structure of communities based on the network topology.**
  - Allows the analysis of a network at different resolutions
- **Approach: Recursively remove the "weakest" edge**
  1. Find the edge with the least strength
  2. Remove (cut) the edge and update the corresponding strength of each edge
  3. Recursively apply the above two steps until a network is decomposed into desired number of components.
  4. Consider each component to form a community.

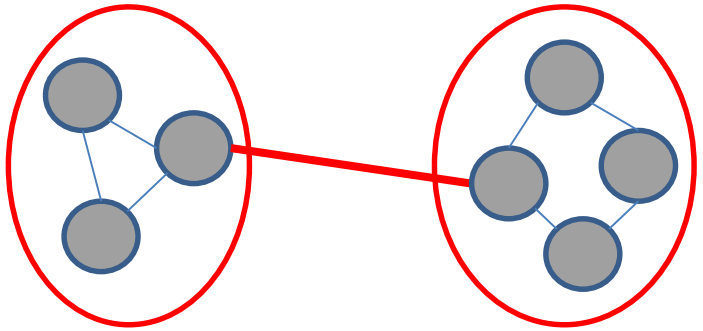
Edge Betweenness

- The strength of an edge can be measured by edge betweenness.
- Edge betweenness: The number of shortest paths that pass through the edge.

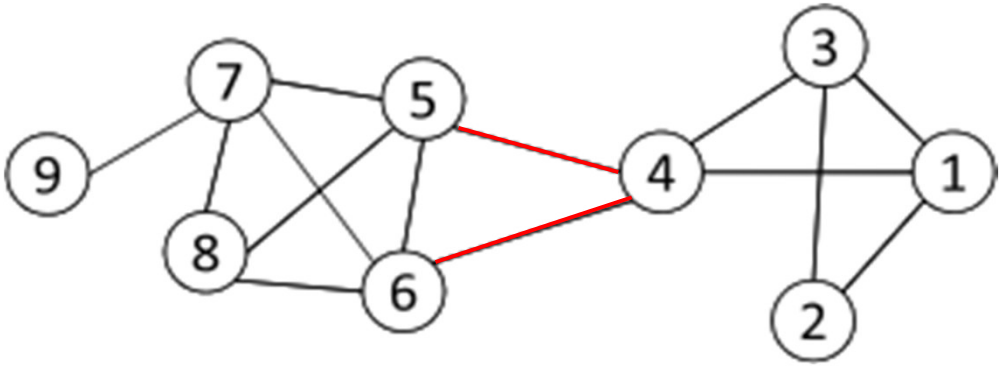


The edge betweenness of e(1, 2) is 4 (=6/2 + 1), as all the shortest paths from 2 to {4, 5, 6, 7, 8, 9} have to either pass e(1, 2) or e(2, 3), and e(1,2) is the shortest path between 1 and 2

- Edges with high betweenness tend to be bridges between two communities.



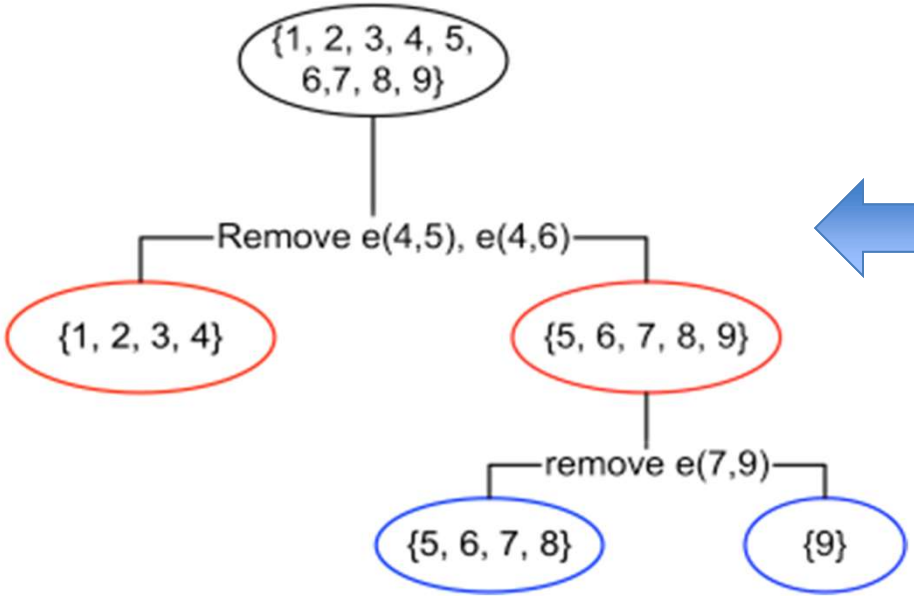
Divisive Clustering based on Edge Betweenness



Initial betweenness values

Table 3.3: Edge Betweenness

	1	2	3	4	5	6	7	8	9
1	0	4	1	9	0	0	0	0	0
2	4	0	4	0	0	0	0	0	0
3	1	4	0	9	0	0	0	0	0
4	9	0	9	0	10	10	0	0	0
5	0	0	0	10	0	1	6	3	0
6	0	0	0	10	1	0	6	3	0
7	0	0	0	0	6	6	0	2	8
8	0	0	0	0	3	3	2	0	0
9	0	0	0	0	0	0	8	0	0



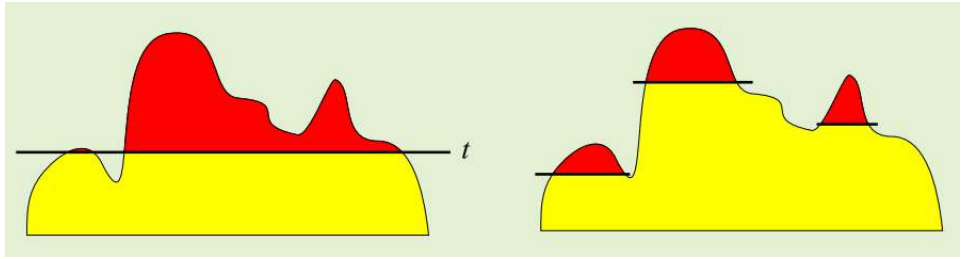
After remove e(4,5), the betweenness of e(4, 6) becomes 20, which is the highest;

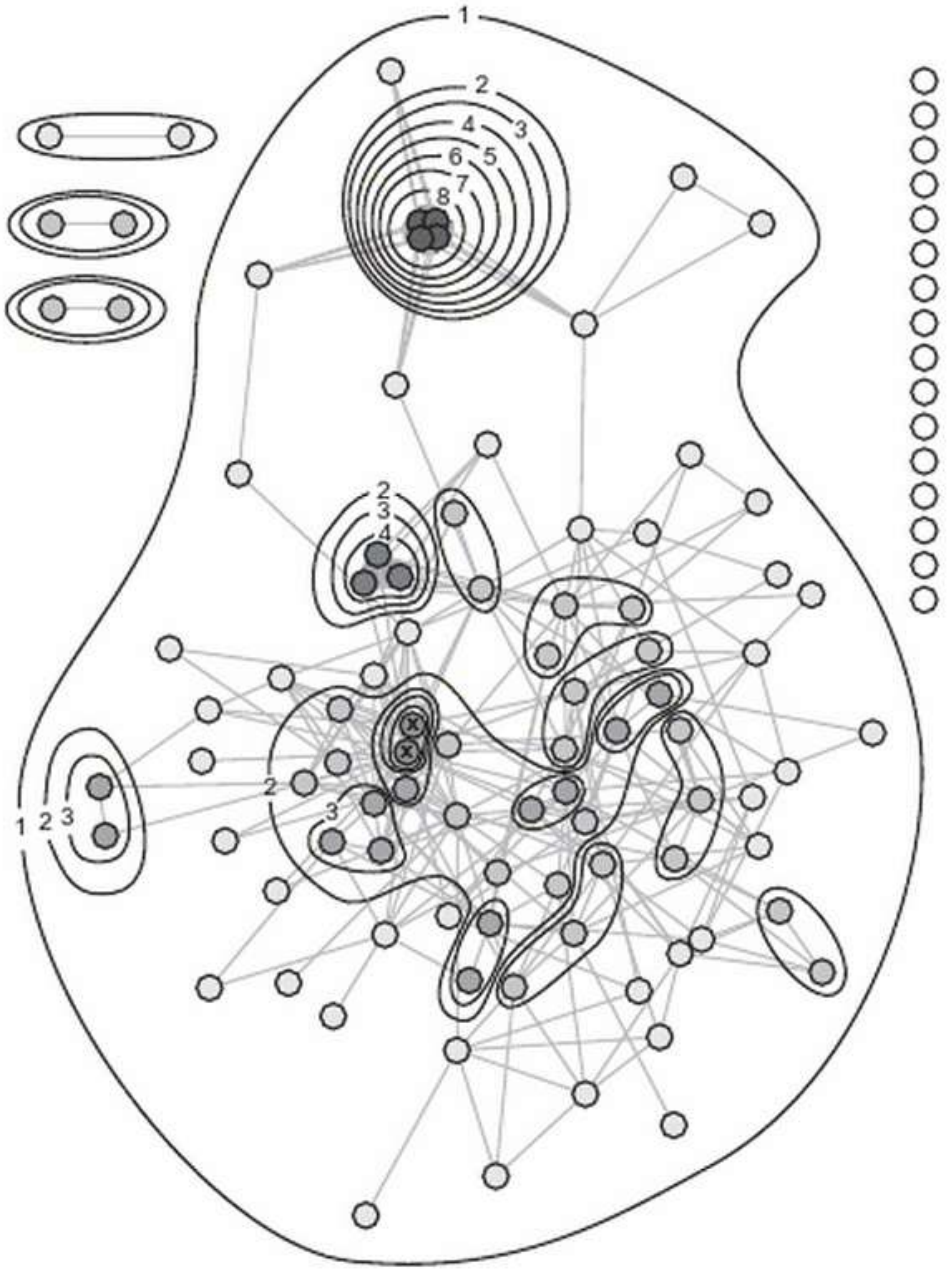
After remove e(4,6), the edge e(7,9) has the highest betweenness value 4, and should be removed.

3.5 Islands

- **Community detection method for weighted graphs, e.g.**
  - number of interactions on a social network
  - number of times author A cited author B
  - overall duration of phone calls

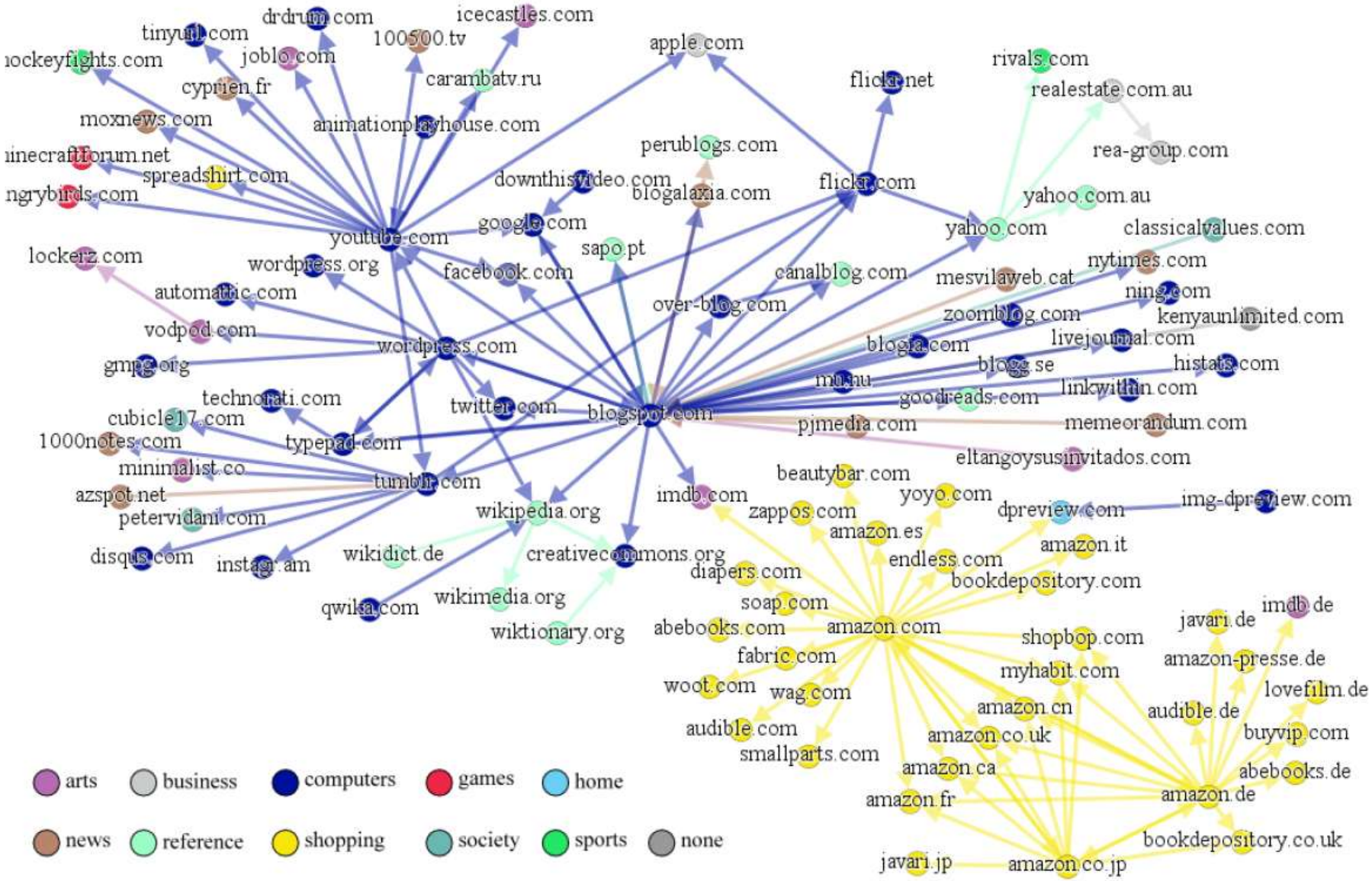
An island is a maximal subgraph connected by lines with a value greater than the lines to vertices outside the subgraph.





Backbone of the WDC Hyperlink Graph

Websites connected by more than 500,000 links. The colors refer to DMOZ categories.



http://webdatacommons.org/hyperlinkgraph/2012-08/topology.html

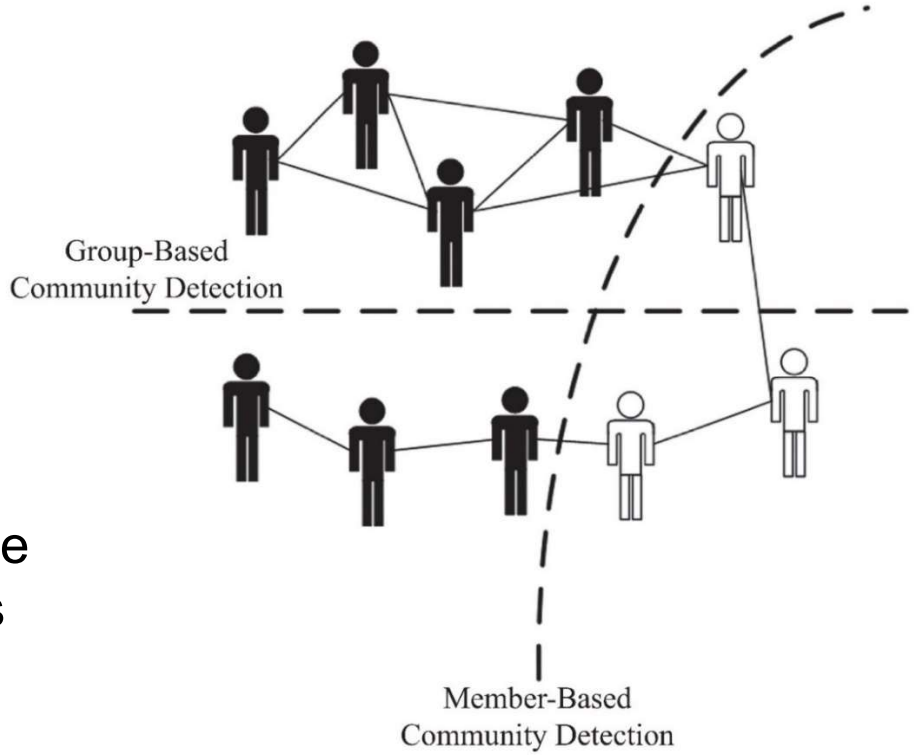
3.6 Communities based on Vertex Attributes

Sometimes it makes sense to form communities based on member attributes and not based on frequent interaction.

- All users that subscribe to a certain channel
- All actors that show a certain behavior, e.g. buy overlapping products
- All persons above a certain age

Group vertices based on group attributes e.g. frequent interaction

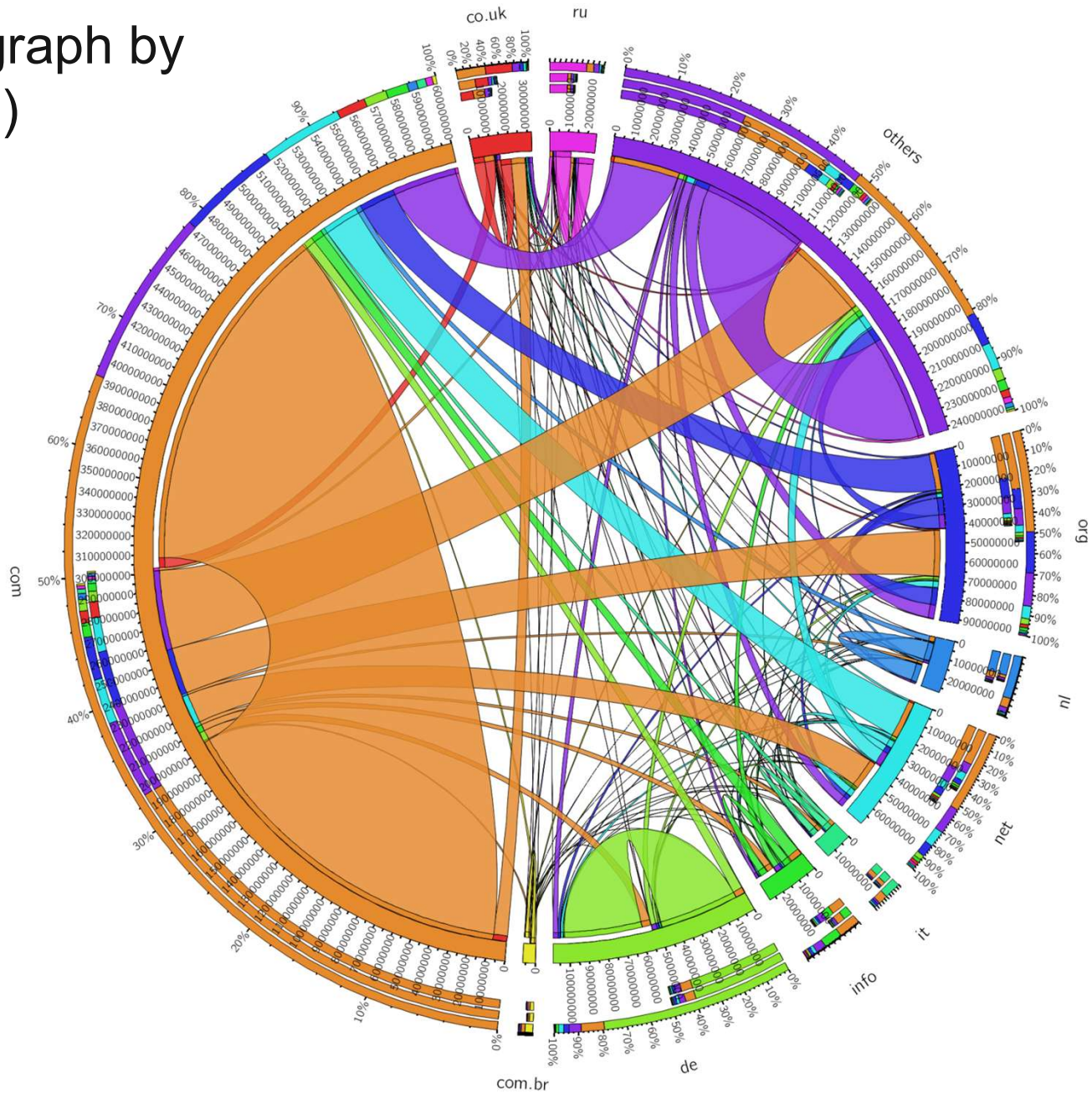
Communities as cohesive subgroups: Our previous definition.



Group vertices based on member attributes, e.g. vertex attribute or vertex similarity

Example: Partitioning by Vertex Attribute

1. Partitioning of the web graph by Top-Level-Domain (TLD)
2. Visualization of the amount of links within and between partitions.



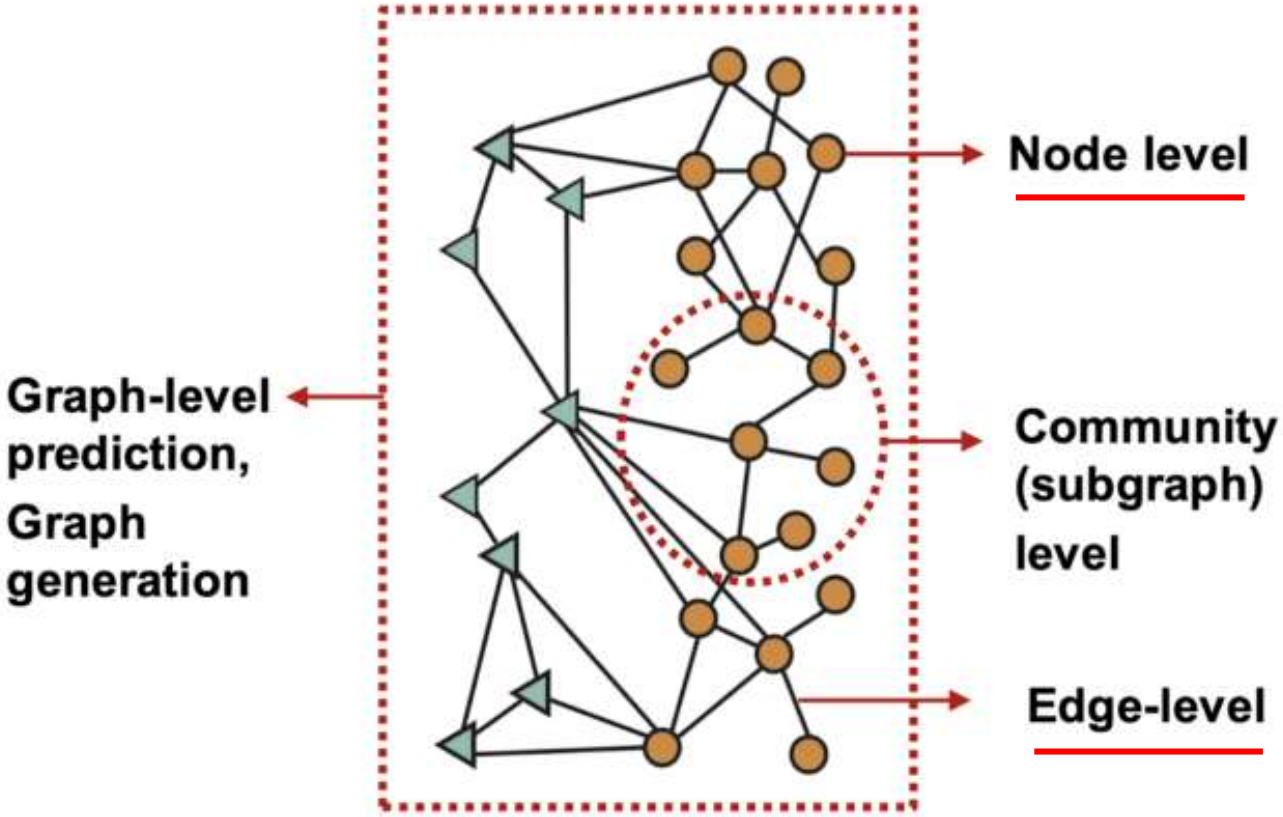
http://webdatacommons.org/hyperlinkgraph/topology.html

Summary: Community Detection

- The different approaches use different features for clustering vertexes:
  1. Components: Vertex connectivity
  2. K-Cores: Vertex degree
  3. Cliques: Maximal density
  4. Overlapping complete subgraphs: high density and bridges
  5. Divisive hierarchical clustering: Betweenness on shortest paths
  6. Islands: Edge weights
  7. Vertex Attributes
- Which feature/approach fits depends on the concrete use case

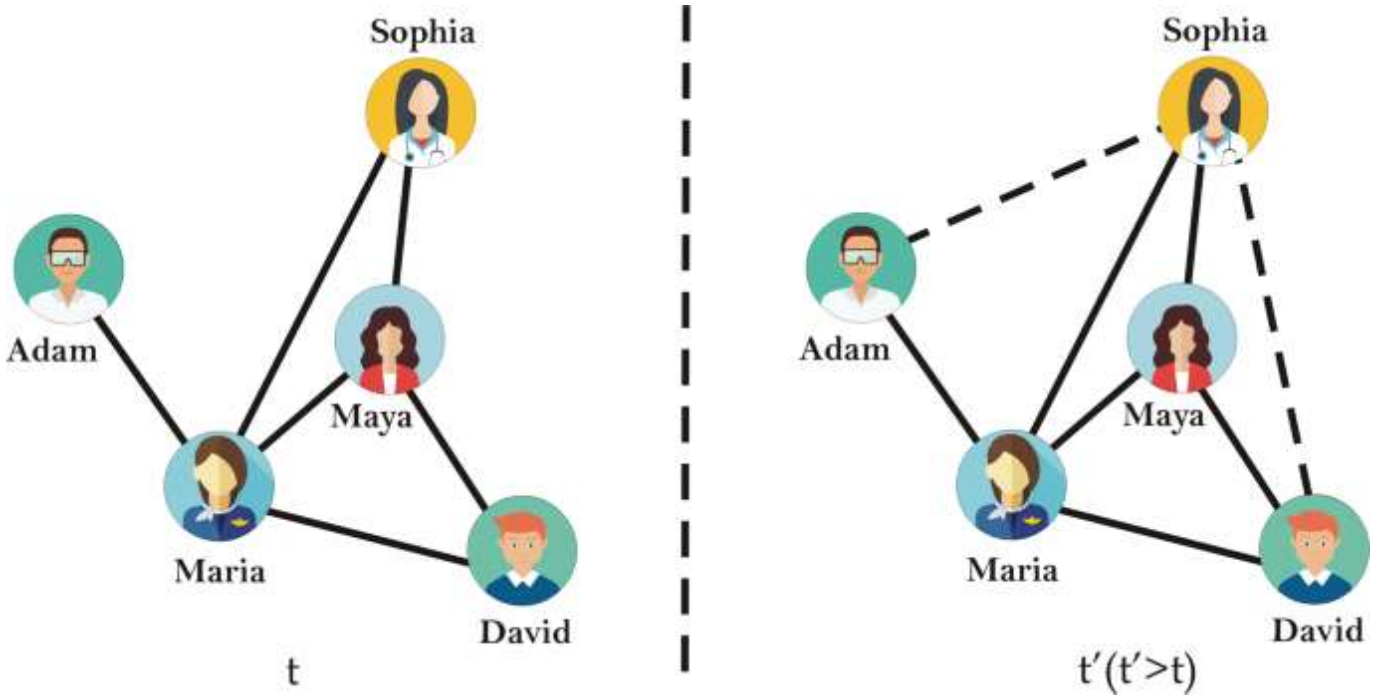
4. Machine Learning with Graphs

- **Goal: Learn a model from a graph for predicting missing edges or node properties**
- **Machine learning tasks on graphs:**



Link Prediction

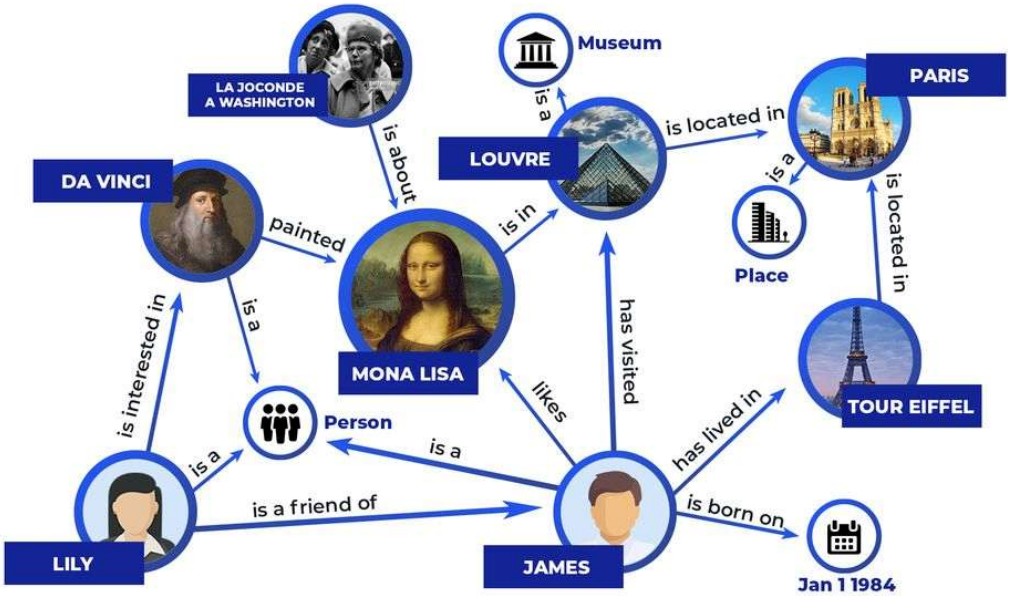
- The task is to predict new links based on existing links

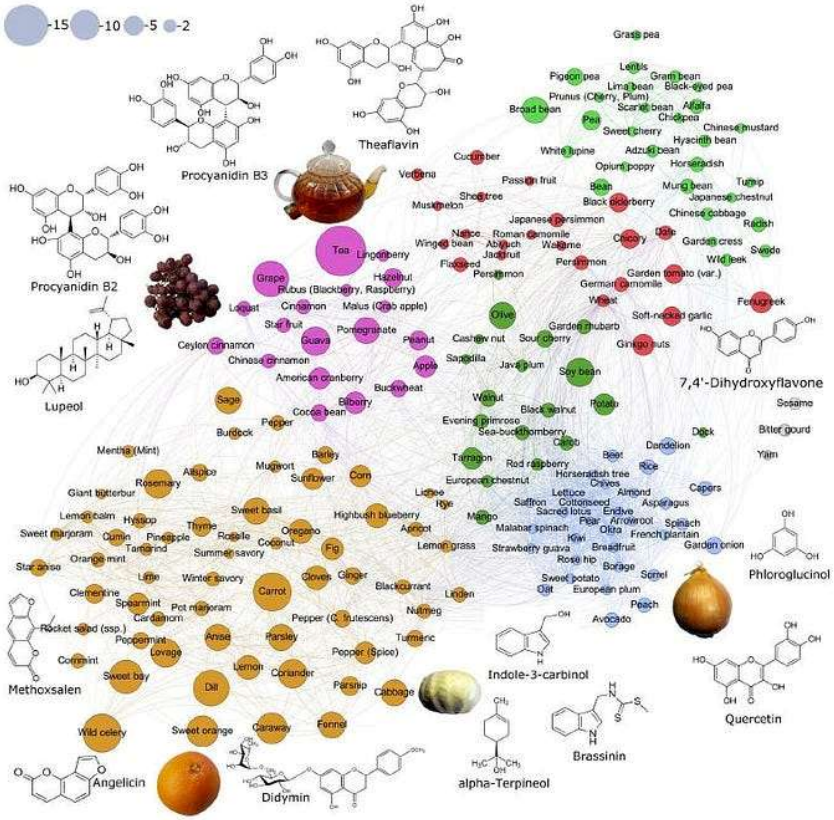


- Applications:
  - friend suggestions on Facebook
  - knowledge graph completion
  - product recommendation

Node Classification

- The task is to predict a vertex attribute value based on its neighborhood

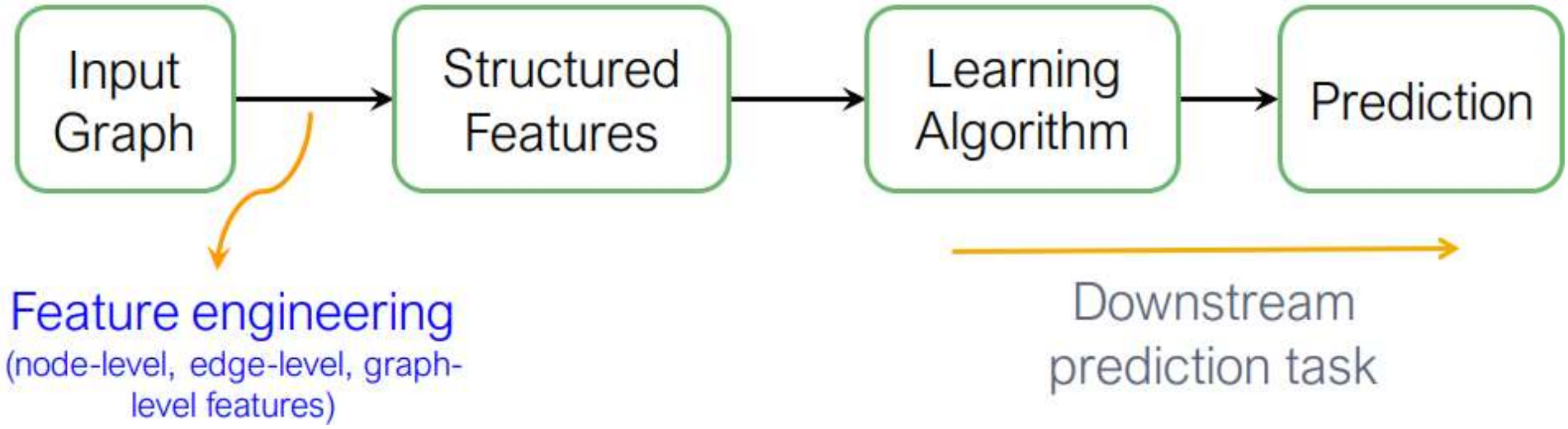




- Applications
  - type prediction in knowledge graphs
  - molecule property prediction: cancer beating?
  - bot detection in social networks
  - topic detection in citation graphs

Feature Engineering

- **Traditional approach: Manually select graph features that you expect to be useful for the task at hand.**

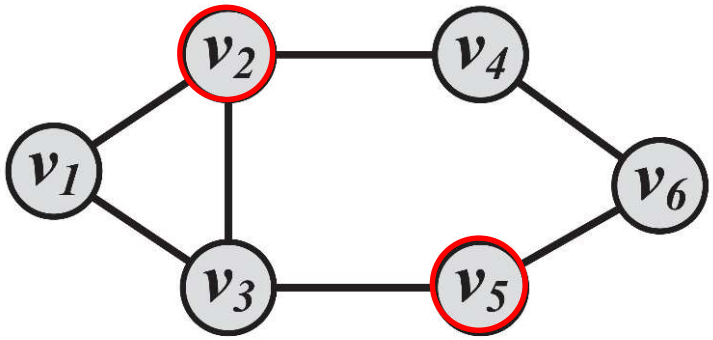


Using traditional learning methods

(Radom Forests, SVMs, Naive Bayes, …)

Feature Engineering

- **Node-level features**
  - characterize the position of a node in the network
    - node degree, node centrality
    - clustering coefficient
- **Link-level features**
  - characterize a pair of nodes
    - distance between nodes, e.g in a categorization tree
    - similarity of the node neighborhood, e.g common friends
  - example: using Jaccard to calculate neighborhood similarity:

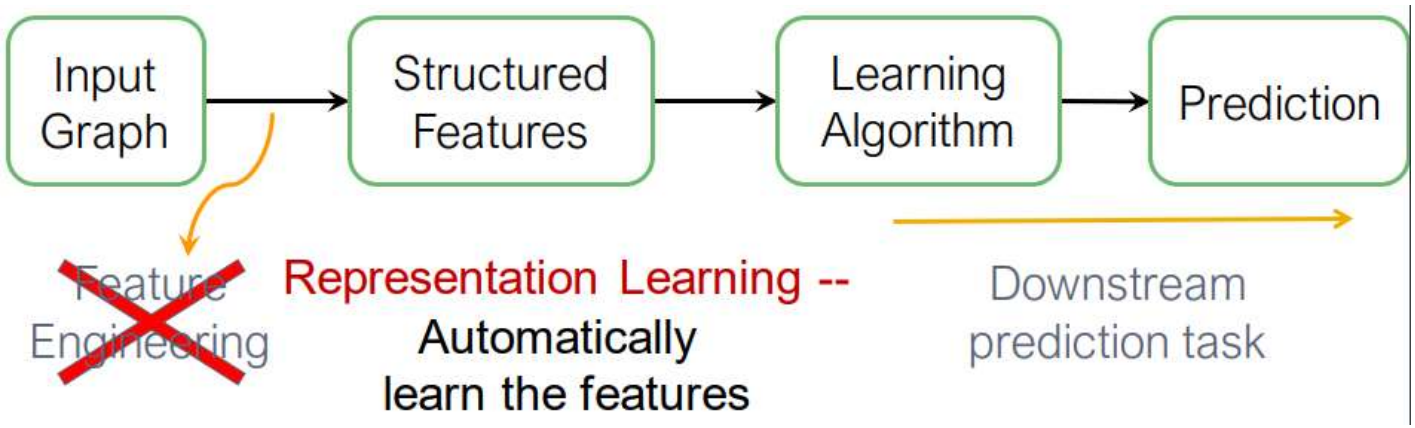


$$\sigma_{\text{Jaccard}}(v_i, v_j) = \frac{|N(v_i) \cap N(v_j)|}{|N(v_i) \cup N(v_j)|}$$

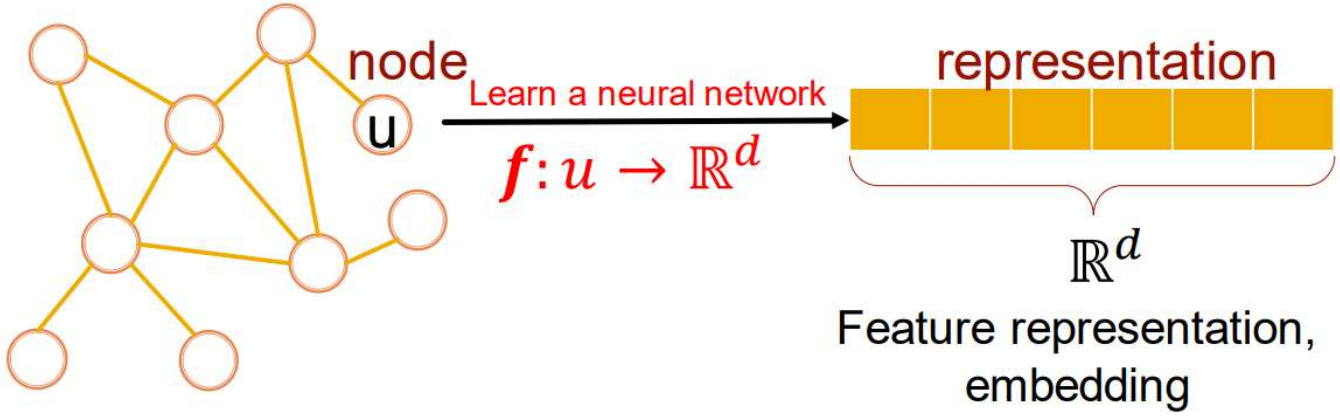
$$\sigma_{\text{Jaccard}}(v_2, v_5) = \frac{|\{v_1, v_3, v_4\} \cap \{v_3, v_6\}|}{|\{v_1, v_3, v_4, v_6\}|} = 0.25$$

4.2. Graph Representation Learning

- **Goal: Efficient task-independent feature learning for machine learning with graphs!**

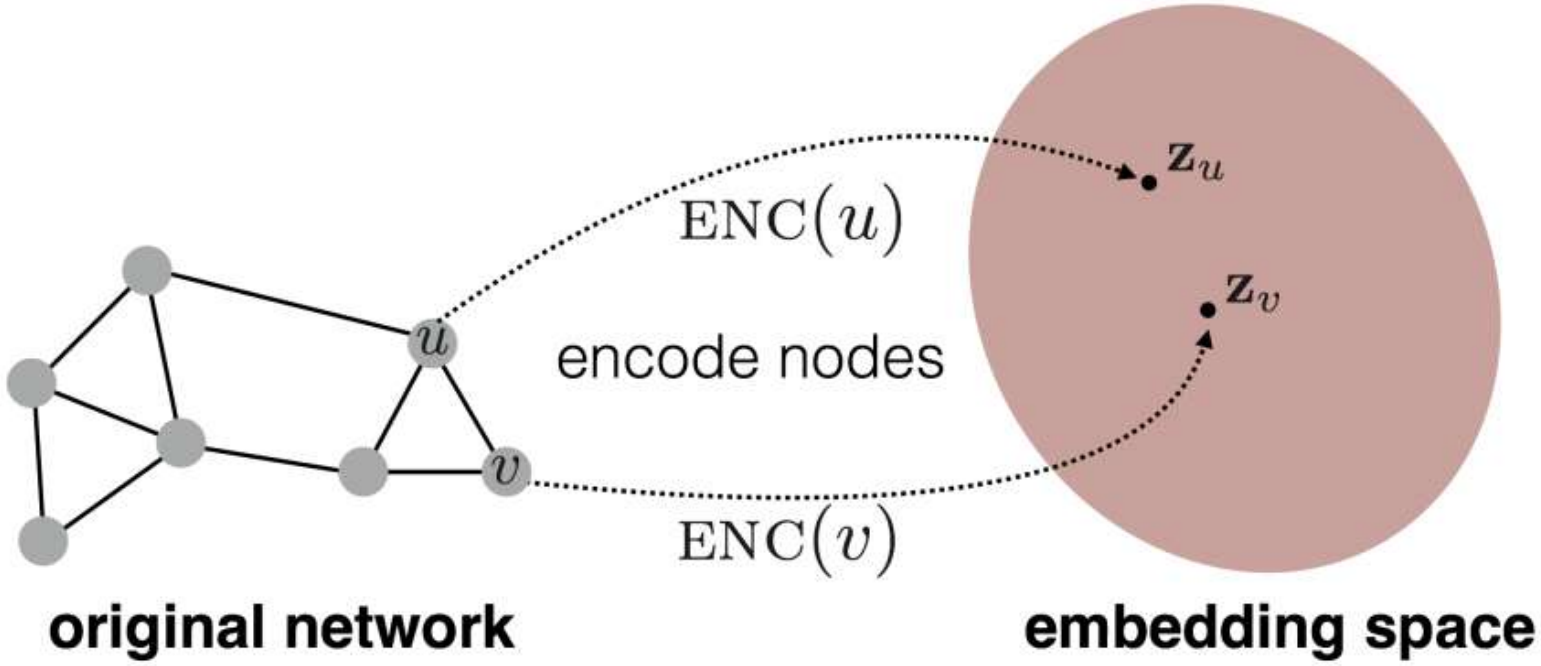


- **Map nodes to d-dimensional embeddings such that similar nodes in the graph are embedded close together**



Learning Node Embeddings

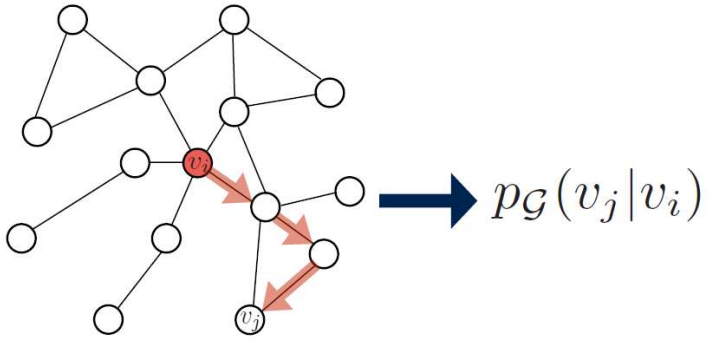
- Goal is to encode nodes so that similarity in the embedding space (e.g., dot product) approximates similarity in the graph



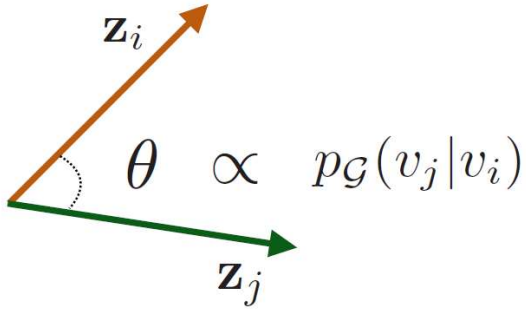
- Function ENC() encodes nodes into embedding space
  - ENC() is learned from the graph

Learning Node Embeddings

1. **Choose a notion of node similarity**
   - **number of shared neighbors or**
   - **random walk-based proximity, used by Node2Vec**
     - do walks to estimate the probability $p_G(v_j \mid v_i)$ to reach node v_j on a short random walk from v_i
     - flexible stochastic measure of node proximity
2. **Optimize the embedding function ENC() using stochastic gradient decent: $v_j^T v_i \approx p_G(v_j \mid v_i)$**



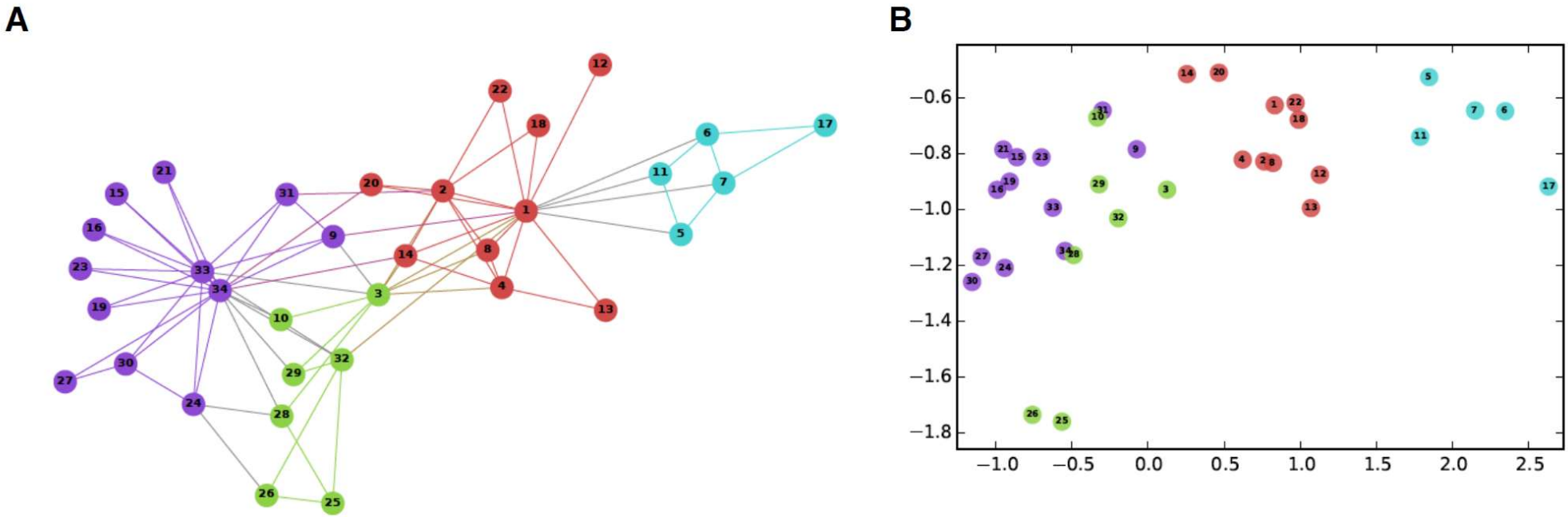
1. Run random walks to obtain co-occurrence statistics.



2. Optimize embeddings based on co-occurrence statistics.

Aditya Grover, Jure Leskovec: node2vec: Scalable Feature Learning for Networks. KDD 2016.

Example of Node2Vec Embeddings

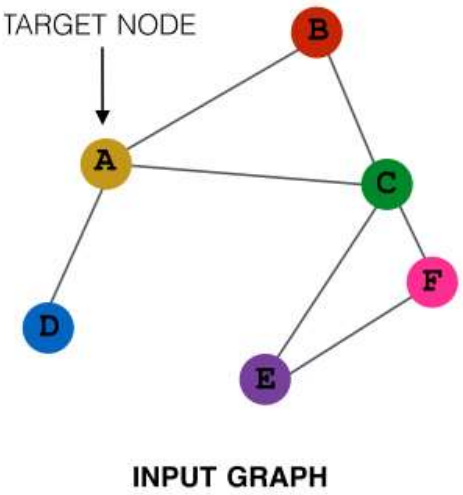


Shortcomings of shallow node embeddings

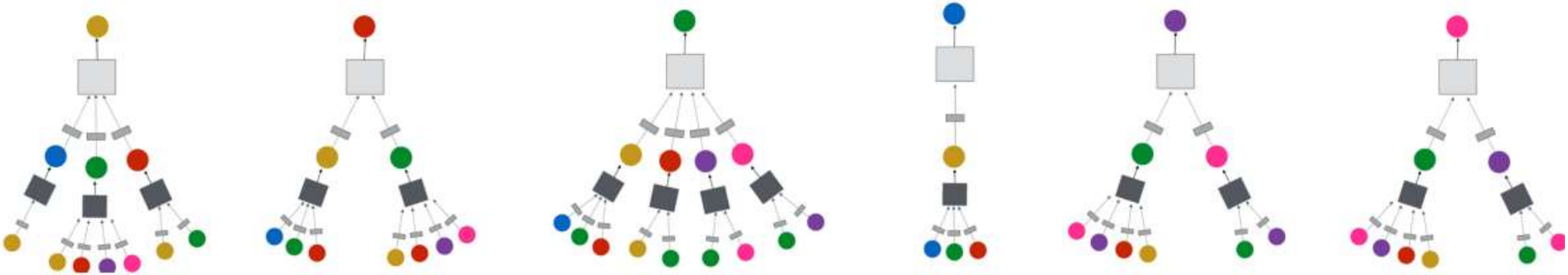
1. do not utilize node or edge features
2. cannot handle unseen nodes as the specific graph is embedded
3. problems with large graphs as every node gets its own input neuron
4. no end-to-end learning as features are learned independent of down-stream task

4.3 Graph Neural Networks

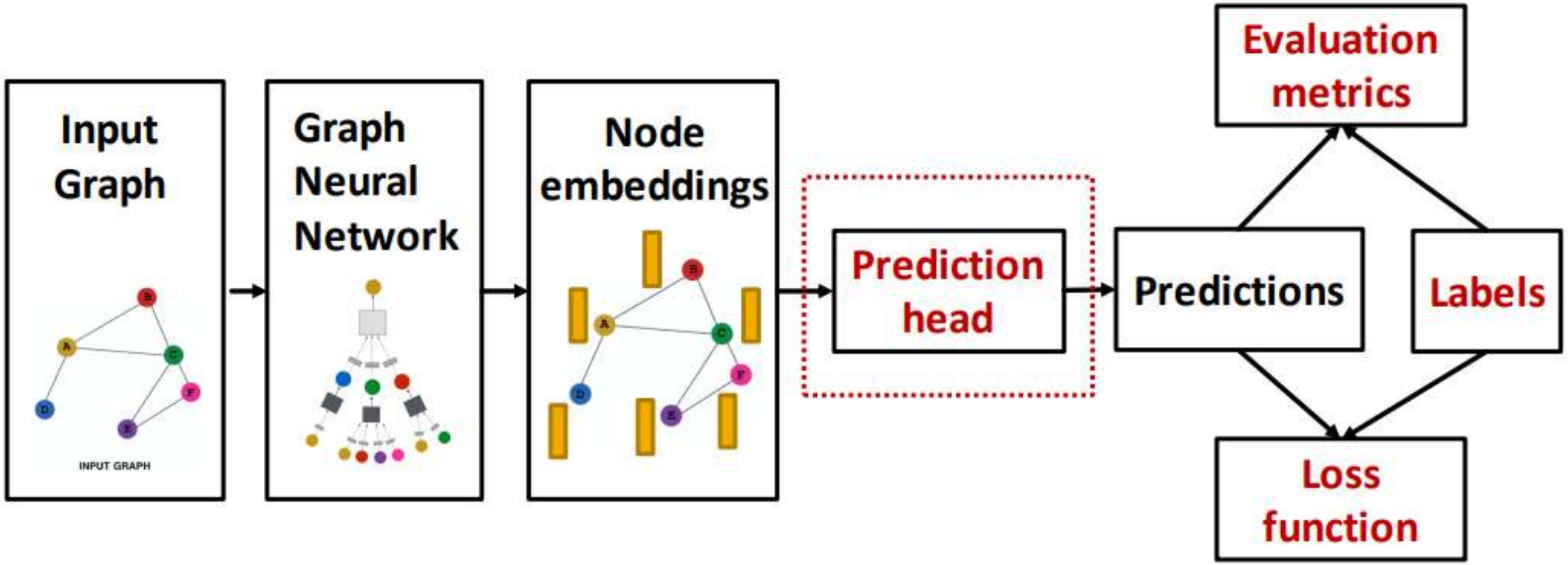
- **Intuition: Nodes aggregate information from their neighbors using neural networks**
- **Each node defines its own Computation Graph**
  - each edge in this graph is a transformation/ aggregation function
  - node embeddings are learned using message passing
  - the transformation/aggregation functions share weights across computation graphs
  - the node features determine initial embeddings



RESULING COMPUTATION GRAPHS



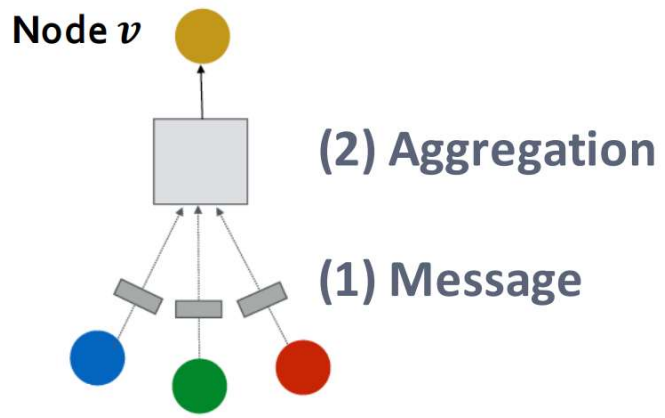
Supervised Training of a GNN



- **Examples of Prediction Heads**
  - Node classification: Linear layer on top of node embedding
  - Link prediction: Dot product of the two node embeddings
- **End-to-End training using Stochastic Gradient Decent**

Variants of Graph Neural Networks

- **Graph Convolutional Networks (GCN)**
  - use simple permutation invariant aggregation functions, e.g. sum(), mean() or max()
  - sum result with embedding of node itself
- **GraphSAGE**
  - subsamples neighbors to speed up computation
  - offers more expressive aggregations, e.g. LSTM
- **Graph Attention Networks (GAT)**
  - aggregation uses multi-headed attention to focus on messages from relevant neighbors
  - current state-of-the-art architecture
- **Implementation: PyG (PyTorch Geometric)**
  - Python library that implements various GNNs
  - https://www.pyg.org/

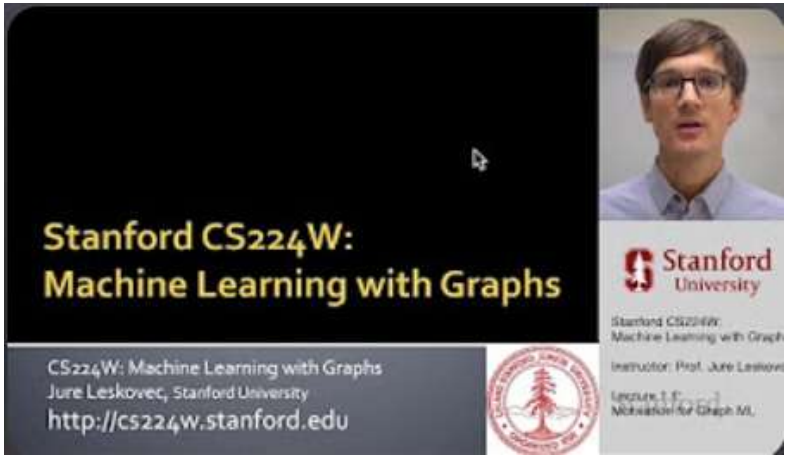


Digging Deeper into GNNs

Stanford CS224W: Machine Learning with Graphs

Local course: Gemulla: IE 678: Deep Learning

- excellent lecture by Jure Leskovec
- Slide sets
  - http://web.stanford.edu/class/cs224w/
- Lecture videos
  - https://www.youtube.com/playlist?list=PLoROMvodv4rPLKxIpqhjhPgdQy7imNkDn
- Student projects
  - https://medium.com/stanford-cs224w



Graph Data Repositories and Benchmarks

- **Stanford Large Network Dataset Collection**
  - social networks, e-commerce networks, citation networks
  - https://snap.stanford.edu/data/

- **Common Crawl**
  - regularly publishes host- and domain-level web graphs
  - host graph May 2021: 515M nodes and 2.82B edges
  - https://commoncrawl.org/connect/blog/

Common Crawl

- **Scientific Network Data Repository**
  - networks from 30+ categories / disciplines
  - https://networkrepository.com/

- **Benchmarks**
  - Node Classification: https://paperswithcode.com/task/node-classification
  - Link Prediction: https://paperswithcode.com/task/link-prediction

Literature

1. Zafarani, et al: Social Media Mining. Cambridge University Press, 2014. Free online version http://www.socialmediamining.info

2. De Nooy, et al: Exploratory Social Network Analysis with Pajek, Second Edition. Cambridge University Press, 2011.

3. William L. Hamilton: Graph Representation Learning, Morgan & Claypool Publishers, 2020. Free online version https://www.cs.mcgill.ca/~wlh/grl_book/